Porting MIIND Extension

Minh Nguyen
BSc Artificial Intelligence
Session 2011/2012

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)_____

Summary

MIIND is a biology neural simulating C++ framework. In order to remove its C++ context and the need for local IT infrastructure, a project carried out the task of cloudifying and porting MIIND to a private university-own cloud in order to design the program as a service which can be accessed via a web-page. This project purpose is to further develop Porting MIIND on the university own-cloud, mainly modifying the input layer interface to make it compatible with handling XML file as simulation input. Another objective is to design and implement a program which allows developer to update MIIND program stored on the cloud which also support the Porting MIIND development.

Acknowledgements

I would like to thank the following for their assistance and support that led to the achievements of this project:

- My supervisor, Marc DeKamps, for providing me with helpful feedbacks and clear guidance throughout the whole project. And thanks to him, a new updated working version of MIIND was constructed quickly, improving the progression of the project.
- My second supervisor, Karim Dejima, for his advices on project design and assistance with contacting other students and experts who are familiar with the tools and software involved with the project.
- The developer of the original Porting MIIND, Bjorn Gerckens, for constantly awnsering my emails, helping me facilitate the implementating processes. Without his constructive advices and suggestions, the project could not have progressed and the development of Porting MIIND would not be so sucessful.
- Django ArmStrong, for supporting me in interacting with the cloud test-bed, linux-shell command, the XEN virtual machines and OpenNebula level images.
- My assessor, Andy Bulpitt, for his detailed and coherent feedback in the Mid Project Report and the progress meeting.
- My family and friends, who have always been supporting me these whole three years at university of Leeds.

Contents

1 Introduction

1.1 Overview

MIIND is a neural simulating C++ framework used in neuroscience for modelling neuronal network. Due to the complexity of C++ and a large amount of computer resource required to execute the simulation. Deep knowledge in C++ and local IT infrastructure are essential. Porting MIIND is a cloudified version of MIIND which has the scope to address the above issues of MIIND, the program was designed as a service on cloud which can be accessed by user via a web browser. As MIIND is constantly developed, new features are added to the program, it is essential for Porting MIIND to be developed to and become compatible with new version of MIIND and take advantage of its new features. Furthermore, both MIIND and its cloudified version are still in early developing stage.
This project aim focuses mainly on extending Porting MIIND, addressing the current drawback of Porting MIIND and implementing applications that can utilize the new MIIND version, specificly its XML usage feature. Another main objective is to design a method allowing developer to update MIIND on the virtual machine. Possible additional objectives involve with improving MIIND performance by parallelisation.

1.2 Objectives

The objectives of this project are

- Objective 1: A literature review on cloud computing, MIIND and the final report of the original project. Understanding the designing and structure of the implementation.
- Objective 2: Retrieve the source code of of Porting MIIND from Bjorn Gercken, study the code and the its structure. Setting up MIIND, update the MIIND version on the cloud with the new version and understand the work flow of the implementation
- Objective 3: Implement a new function for Porting MIIND, which allows XML files as simulations inputs and modify the program to handle the new type of input.
- Objective 4: Implement the auto-update function which automatically update MIIND on the cloud

- Objectives 5: Evaluation and testing on the implementation, comparing the original Porting MIIND and the new version. Suggestion of possible extension, a review on what have been done and could there have been an alternative solution.

1.3 Minimum Requirements

The minimum requirements are:

- Modify Porting MIIND to allow XML file to be used as simulation input.
- Implementing an auto-updater program which can update MIIND without requiring any human-interaction.

1.4 Deliverables

- Deliver the new Porting MIIND implementation which can retrieve and handle the XML files as MIIND simulation input.
- Deliver the MIIND auto-update application which can automatically update MIIND program on one of the virtual machine of Porting MIIND design.

1.5 Possible extension

- Implementing a parallelisation version of MIIND.
- Redesign the XML implementation, allowing user to submits multiple XML files in one request

2 Schedule and Project Management

2.1 Original Schedule

This original schedule was created on week 5, based on the amount of work available and the current progress of the project, detail on chart figure 1.1 . Due to the nature of the project is extending an existing software, most of the early times were planned on preparing for the actual implementing tasks. The first few weeks were intended to be used for literature search about relevant areas to the project and studying the software (this include, setting up, updating and testing the software). The coding parts of the XML implementation and MIIND updater were seen as simple tasks which expected to require a small amount of time to proceed and planned to be proceeded during Easter holiday. Research on parallelisation of MIIND was also intended to be carried out during the holiday, afterwards implementation of parallelisation MIIND was expected to take about 1 month to complete as the task was described quite complex and difficult. The last month before the deadline was planned to be spent on project writing and

evaluation of the softwares. Milestone was set based on the type of objectives, this was to ensure that project could be progressed in order.

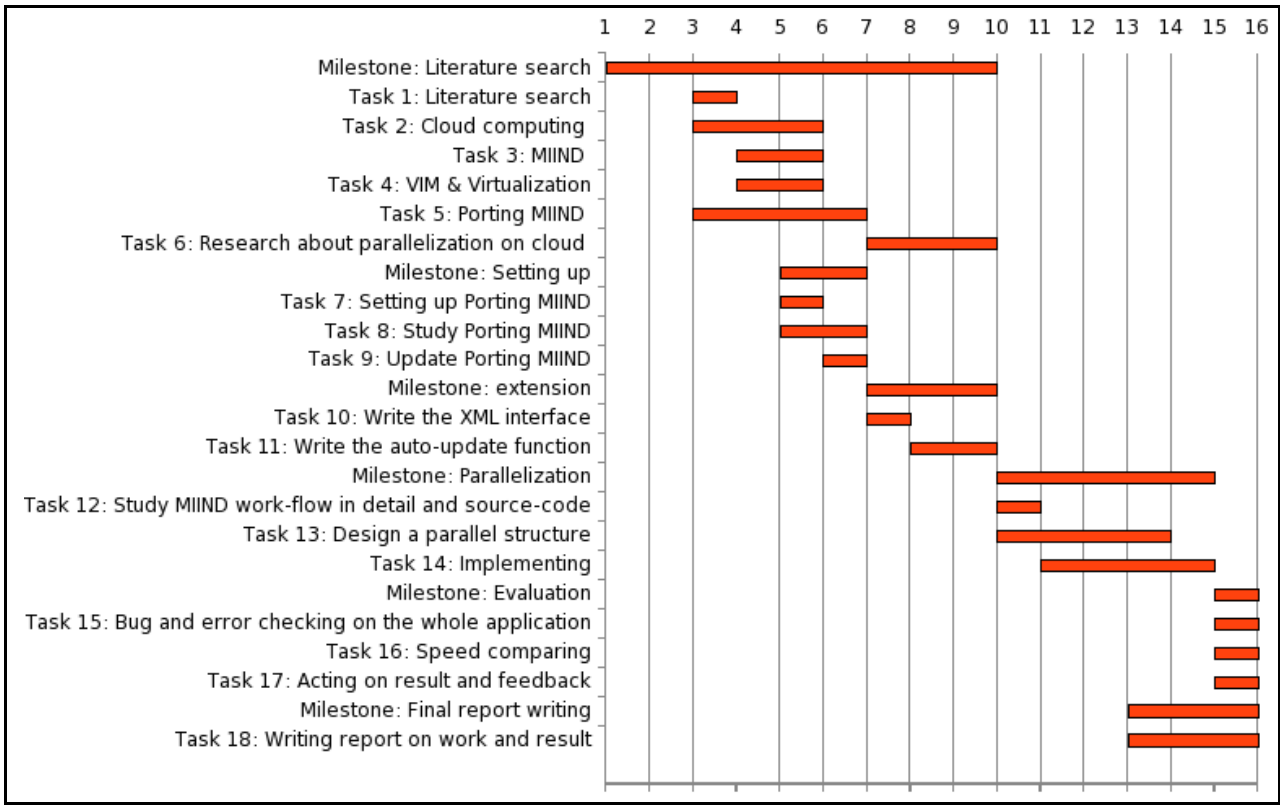


Figure 1: Original Schedule

2.2 Revised schedule

The whole schedule was forced to change due to some tasks being underestimated and the milestones were planned without any consideration of other factors (gathering information and data, issues with third party software, ...), these required the project schedule to be revised frequently. To catch up with the schedule, all implementation tasks were proceeded altogether, so that when one was not progressing, the others could be worked with instead to avoid wasting time. The progression of the XML implementation was very slow and unproductive; the minimum requirement was finished at a very late stage of the project. On the other hand, the MIIND-updater implementation was carried out well and without any major issues, the final code of the MIIND-updater was finished 1 week before schedule. Due to the delay caused by slow progression of XML implementation, the extension objective; Parallelisation version of MIIND was removed from the schedule. The final schedule was planned on week 9 based on the current progression. Detail on chart figure 1.2.

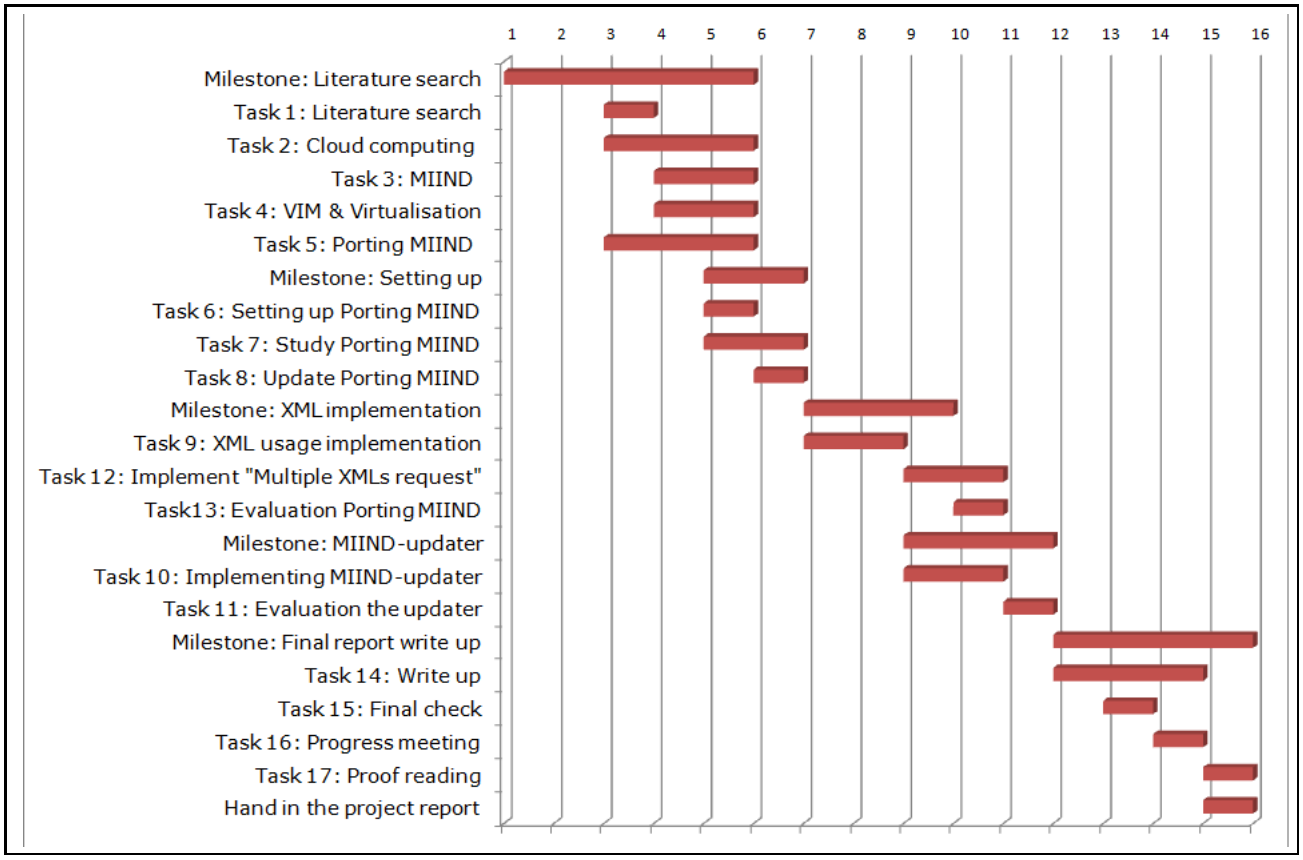


Figure 2: Revised Schedule

2.3 Relevance to Degree Programme

This project is mainly relevant to module Distributed Systems (TC30) in architecture and design. The actual implementation involves Software Systems Engineering (CR21) Java programming knowledge. The application of experimenting, MIIND is related to Biologically Inspired Computation (AI33). Some of the researched materials were from module Parallel Scientific Computing (TC32).

3 Background Research

3.1 Overview on Cloud

This section is a brief review about why cloud computing was chosen for the implementation of Porting MIIND

Distributing systems usually provide three main benefits: high performance and storage capacity that surpass a single machine and the remote access to the facilities that provide the former two [17]. Out of many model and design such as grids, cloud computing, data-centre, web-services and RMI/RPC, despite all of them can satisfy the requirement to implement Porting MIIND, cloud was chosen for the implementation. Detail of comparing cloud with other models are given in original Porting MIIND report, section 2.2. [18]

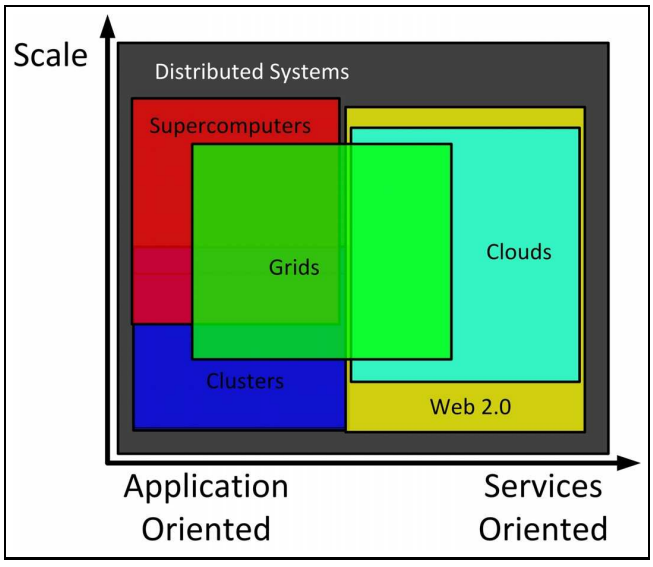


Figure 3: Grids and Clouds Overview.[18]

The two most compatible designs for Porting MIIND were grid and cloud. Both seem to meet all three requirements on a acceptable level. They closely resemble each other, hold the capability of providing storage and computing resource. However, cloud is considered an evolution of grid,

it is a form of service oriented paradigm providing abstract resource to its user rather than grid where users are member of the grid network itself. [18] [17]

3.1.1 Why cloud was chosen?

The design of grid is rather decentralised. In 2002, Grid was described by Ian Foster [14] as "a system that coordinates resources which are not subject to centralised control, using standard, open, general-purpose protocols and interfaces to deliver non-trivial qualities of service". It is a distributed computing paradigm that used by multiple virtual organisations (VO) that has contribution or relate to the group. The resource distribution is in a form of fair and share to VOs and applies to a common task while cloud assigns resource to its user. [17]

Cloud was centralised architecture until recently, it is now both decentralised and centralised as nowadays, database is not stored only in one place, but duplicated on different virtual machine. [19] Due to this, cloud is again a better choice, we can dynamically modify which part is centralised, this is a suitable design when dealing with MIIND output resource and libraries but still separates the main application on a different virtual machine.

Cloud is also much more scalable and flexible, it provides service only as much as demand, no more or longer than the requirement, this is essential when develop a software service, user only need to pay for what they use just as a public utility service of electricity, gas or water, this feature of cloud is a business model which was referred to as utility computing [22] . Grid on the other hand, is stricter and put schedule on usage. In most case, in order to get time and usage of grid resource, a grid member must also contribute to the grid as well and received the scheduled time to use other resource. [17] These show that cloud is more suitable as Porting MIIND was implemented with the ideal of acting as a web-services provided to any user who has interest in neural network simulation, which are mostly neuroscientists.

Furthermore, cloud has different service models: Software as a service (SaaS), user can pay to use certain software and application run on the cloud without requiring license or registering for the software, there is no need for the user to update the software or maintain it. This approach is used by many company such as, Amazon, Yahoo, Googles, ... As users who run MIIND are sometimes non-technical, this approach can render some of MIIND drawbacks, creating an user-friendly design. Platform as a service (PaaS), user can rent hardware, operating systems, storage and network capacity over the Internet platform for running existing applications or developing and testing new ones [20]. Infrastructure as a service (IaaS) is considered similar to utility computing but with the addition of computing virtualisation [15], A large set of resources are controlled by the infrastructure provider, they can freely resize and assign to match their customer demands which are usually software provider [21]. Cloud IaaS is sometimes seen as

the combination of utility computing and grid computing. This service model was used for storing data output and libraries of Porting MIIND. [15] [23]

3.2 A brief overview of MIIND

Cognitive neuroscience has been making great progress in experimentation, but to achieve further development and better experiment, understanding specific part of the brain seem to be the appropriate step [16] . This comes to the need of a good modelling and presentation of brain activity. However, the model will need to contain high detail and realistic feature of the brain activities. When a model is built, it is important to ensure its being reusable, avoid massive duplication programming in small research group. The model should also have the capability of extending to a more detail level of representation and the form that can be used as a frame to build more complex models. With these functions, a small research group can develop high level cognitive network modelling and transfer the model between different research group and individual. Obviously, these models also need to be evaluated and validated by a researcher in the field, and the developer must have knowledge of cognitive neuroscience. MIIND (Multiple Interacting Instantiations of Neural Dynamics) developed by Dr Marc DeKamps is one of the product that includes the above attributes. Section 1.1. [16]

3.2.1 What does MIIND do?

MIIND (Multiple Interacting Instantiations of Neural Dynamics) is a highly modular multi-level C++ framework. It can be used in various ways. As a simulator, MIIND can model a population of neurons (a network), it is however not used to simulating individual neurons. There are a few reasons for this, modelling the network on at the population level usually appears to be more computationally efficient than processing at individual level of each node. To simulate an individual neuron requires a large amount of computation operations. But in a network, the number of neurons is usually large, modelling the activities of all the neurons leads to a huge amount of computations operations. However, the activities of a population of neurons can still be simulated on population level, skipping unnecessary variables that are required when modelling an individual neuron, this also makes complex network simplified. SectionMiind as a neural simulator [7]. Being a framework, developer can build customisation network using MIIND for network visualisation, serialisation and representation. [7]

3.2.2 Problem and drawback of MIIND

MIIND requires a large CPU and memory resource to process heavy load computations, this presents more issues as MIIND is a serial application designed for running on one single machine. When the simulation runs, ROOT data is generated (serialise) and then visualised in analyse components by creating image.

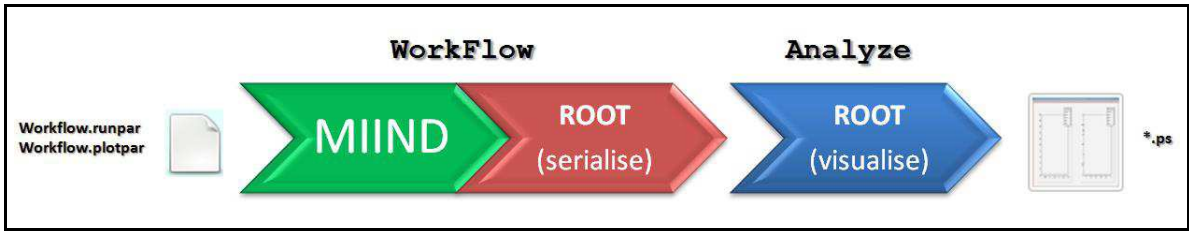


Figure 4: MIIND workflow [18]

Moreover, the framework libraries and output data also takes an amount of storage, putting a large amount of computation and memory used on the computer. All of these lead to the high demanding on local machine infrastructure. Section 5.3 [18]

3.3 An overview on Porting MIIND

3.3.1 MIIND as a service and its benefits

For the purpose of removing MIIND C++ content, MIIND was ported on cloud and implemented as a service that user can access via a web interface. The libraries are all set up on the cloud; user does not have to go through the process of installation of MIIND which is complex and not suitable for non-technical users. Every new simulation does not require re-compiling the C++ programming but instead, neuroscientists can just input the parameter of configuration through the web-interface, this was achieved thanks to a customised version of MIIND developed by Dr Marc Dekamps, moreover, as the simulations are run on the cloud, users can still progress with other tasks on their local computers. This architecture renders the requirement of C++ programming experience and knowledge of libraries set-up unnecessary. Neuroscientists only need access to the cloud and input their configuration via the website to run a simulation, less time consuming and complex. The flexibility and scalable cloud design allows more dynamic allocating resource needed for each user, the virtual machine manager assigns user according to the demand and isolates each user pool of resource, giving an abstract design and makes users unaware of each other. The cloud design also makes sure every user proceeds under the same environment which replicates the simulation and its output, making experimenting and studying easier. In Porting MIIND project, the university own-private cloud was used as test-bed, however when use on actual business cloud like Amazon EC2. The user will need to pay for the computer resource although MIIND is a free application. Section 2.8 [18]

3.3.2 MIIND development as PaaS

MIIND itself is an application to support experiment simulation and studying neuronal network, it is constantly updated with changes and extensions, thus, a developer needs access to the

framework stored on cloud and all libraries of MIIND. As the MIIND framework is stored on a virtual machine, developer can access the cloud and develop the application without using their local computer resource. This model can be seen as Platform as a Service design, it provides the developer an environment with the framework and all libraries needed, developers do not have to worry about the lack of storage or computation resource, as the actual testing and processing use the virtual machine resource. In the existing implementation, PaaS was considered essential due to MIIND being a libraries tool-kit before SaaS could be implemented. Section 2.9 [18]

3.4 Architecture of Porting MIIND

This section describes the architecture and structure of how Porting MIIND is implemented on the cloud.

3.4.1 Layer design of Porting MIIND

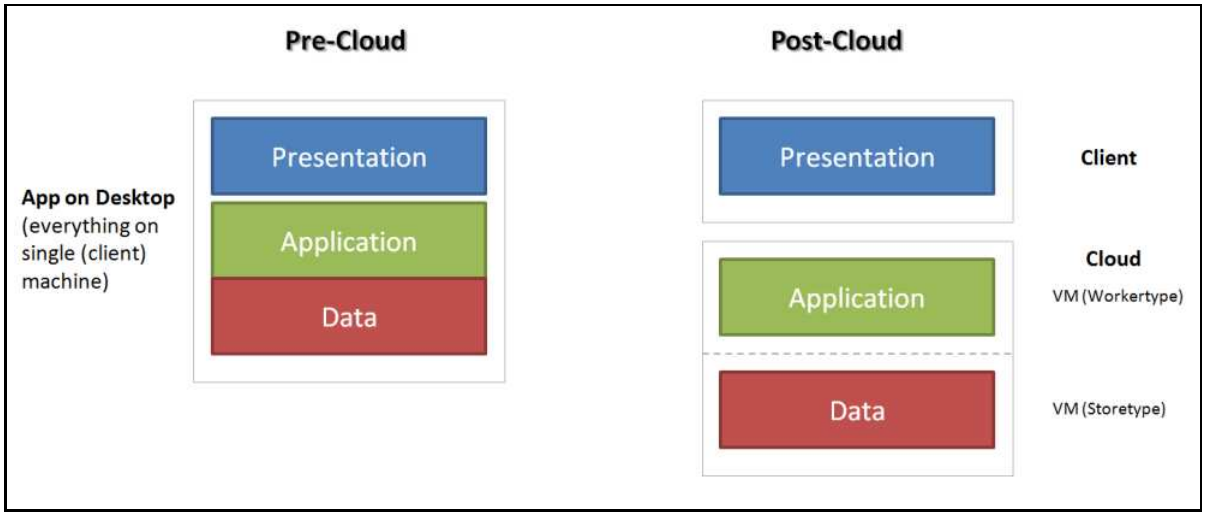


Figure 5: Porting MIIND layer design [18]

As described in MIIND overview, the nature of MIIND is to run on one computer, components and processes are build to run in serialisation, true parallelism is difficult to achieve. However, the customised version of MIIND manages to distribute and separate components of MIIND making the program more flexible to manage. In more detail, original MIIND is designed as 1-tier application where everything (the presentation layer, the application layer and the data layer) is stored and runs on one single machine. The existing implementation of Porting MIIND changes MIIND from 1 tier architecture to 3-tier, presentation layer is a web interface where users input their configuration parameter to run the application (the simulation) that is put on a virtual machine on the cloud, the output data is stored on another virtual machine. 2-tier design (where data layer and application layer can be stored on the same machine) was not used

when data access availability was taken into account. By separating data and application layer, more computer resource and storage is available with simply creating a new virtual machine, this benefit allows data source to be stored on a dedicated virtual machine, resource can be accessed any time for reusing purpose. The application layer contains the customised version of MIIND. Section 5.3 [18]

3.4.2 Virtual machines managing: Xen and OpenNebula

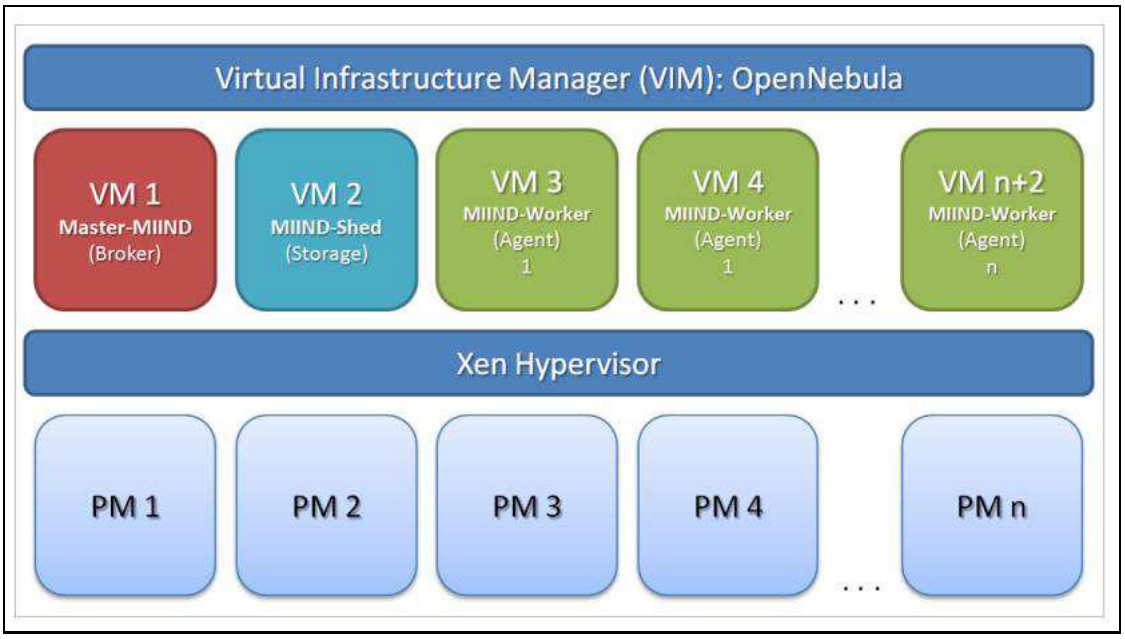


Figure 6: Xen and OpenNebula design [18]

Xen Hypervisor is used as the virtual machine manager; it controls and implements the lower level of virtual images which are the blueprint of the image instances that initialised by OpenNebula. Changes made to the virtual machines on Xen level are permanent and affect on every instance image of that virtual machine. [10], [9]. On the other hand, images of the virtual machine generated by OpenNebula is a temporary instance of the actual virtual machine, change made to this instance does not affect the content of the virtual machine, modifications are non-permanent as the instance is discarded when the image on OpenNebula is shut down. OpenNebula uses two templates to initialise the virtual machine: Image template and virtual machine template. Image template is used to register with the virtual machine on Xen level and create a copy of it in OpenNebula repository, Virtual machine template uses the copied image in this repository and initialise the virtual machine image with preferences memory and resource setting. This process can be skipped and modification would be carried out directly on the original Xen virtual machine. But this come with the risk of alternating the origin and if

there is no backup, in case of corrupt or bug, the virtual machine content is permanent changed. Debian image was used as virtual machine to install MIIND. Section 7.2 Porting MIIND report [18], originally cited from OpenNebula document [8].

3.4.3 The components of Porting MIIND

There are seven virtual machines: Master-MIIND, MIIND-Shed, Web-MIIND, MIIND-Worker, MIIND-Giant, MIIND-PaaS and MIIND-Client. However, only the main four virtual machines are described in this section.

MasterMIIND This is the main component that interacts with OpenNebula images to manage other virtual machines. Master MIIND contains NTP (Network time protocol) to synchronise the time of other machines, managing the number of simultaneous simulations can be run, it also configures the setting for the whole Porting MIIND network (IP range for virtual machines used, location of MIIND-Shed and Web-MIIND and boot up mode for the application). A log is also generated when Master-MIIND is booted up to keep track of the work flow for debugging and monitoring purpose. Section 7.4.3 [18]

Web-MIIND Wed-MIIND is where users requests are first handled and validated. A Tomcat servlet is used to display information to user. By deploying the war file (archive extension of java files) on Tomcat, a web form is created for user to input configuration parameter for MIIND simulation. The web interface also requires an user name and password to use the application, this information is stored in a SQL database which contains simulations associated with their ID and users. There are two validation, a JavaScript validation on the html web-page to make sure user does not enter invalid values and a Java validation on Web-MIIND server to reconfirm that all parameters value are sent correctly and valid. Section 7.4.4 [18]

MIIND worker This is the virtual machine where the actual simulation is run. It receives input and setting values from Master-MIIND. The simulation result and outputs images (which are generated in postscript format and can be converted into another extension base on the setting preference), depend on user request, might be achieved and sent to both Web-MIIND and MIIND-Shed. MIIND-Giant is a MIIND-Worker with large storage and high computing power for handling intensive simulations. Section 7.4.5 [18]

MIIND-Shed The virtual machine where all output data archived sent by MIIND-Worker are stored. It creates a layout folder for each user, inside each folder, there are sub-folders contain simulation data differed by their session ID. The data on MIIND-Shed is permanent and remains even after the OpenNebula image is shut down. Section 7.4.5 [18]

4 Problems and solutions

4.1 XML usage implementation

4.1.1 Description and brief solution

The web-interface of original Porting MIIND only allows user to submit several simulation parameters. This limits the extent of simulation configuration an user could made and the capability to run and create different simulations which requires change in other input parameters that are predefined in the old version of MIIND. To address this issue, a new MIIND version was built which employs the option of using an XML file as the simulation input. All simulation configuration parameters are included in this XML which allows user full control in creating a simulation, the approach of wraping everything in a XML file also makes it fast, simple and convenient when user attempts to duplicate result from previous configuration as each simulation input setting can be save as individual XML file, this is certainly useful when user wish to transfer their XML configuration for testing and studying by other neuroscientist. The main first task is to reimplement Porting MIIND, making it compatible with using XML file as simulation input parameters.

4.1.2 Implementation approach

The task concerns with changing how each layer handles the input parameters (now is an XML file). Firstly, the new version of MIIND needs to be updated on the MIIND-Worker virtual machine. My first objective concerned with reimplement the Web page GUI, then move to the input handling processes on each virtual machines. Instead of typing in the simulation parameters, users just need to submit the XML file, the Web-MIIND virtual machine should then validate the XML file and transfer it to the MIIND-Worker via the Master-MIIND. The second objective is to reimplement how MIIND-Worker handle the simulation input with the new updated version of MIIND.

4.1.3 Evaluation and testing

This task can be evaluated on two aspects; does the programme work properly (the output should be corrected base on input) ? Is there any risks, bug or error ? (This requires test run with different settings and environments). The application can be tested by user from Bio-Computing research group members to get feedback on the user-friendly aspect.

4.2 Multiple XML requests

4.2.1 Description and brief solution

This was an optional objective to extend the XML implementation. Considering the cloud architecture, storage and computer resource available are usually quite spared, this means the simulation can be processed faster and larger amount of data output can be stored. To further make use of these two advantages the cloud provided, I decided to implement a new design which gives users the option to simultaneously submit several simulations. This also serves the purpose of building an user-friendly service, user can request many simulations without repeatedly submitting the input files, saving time and effort of making multiple requests to server, the communication and traffic cost between Web-MIIND and Master-MIIND are reduced and optimised as well.

4.2.2 Implementation approach

This extension can only be carried out after finishing the XML usage implementation. The most suitable design would be simply using Master-MIIND to distribute the request to several MIIND-Workers. The user sends the XML files to server, for each XML file, a MIIND-Worker virtual machine is booted up to carry out the request..

4.2.3 Evaluation

The evaluation is carried out along with the XML implementation testing. Observing the time taken to complete one user request and increasing the number of XML input files. It is also a good idea to compare the performance time between this new version and the original XML implementation.

4.3 MIIND auto-updater

4.3.1 Description and brief solution

As MIIND is stored in a virtual machine on cloud test-bed, developer can make use of remote computer resource and storage for developing and running MIIND, this is PaaS approach, which removes the requirement of local IT infrastructure. However, much to its benefit, this approach also has its drawbacks. Being stored on the cloud accessed by internet connection, this model also inherits the disadvantage and possible problems of internet. The developer might not be able to connect to the cloud due to many reason such as power outage on server side, hardware breakdown, software bug, network interrupted, low speed network, overload on bandwidth, authorisation and sometimes, problems on local machine. Not to mention, sudden connection lost while modifying the code will result in all change and update being lost.

So an alternative approach is abandoning the PaaS approach and goes back to the traditional route. Developer modifies and tests the application on the local computer, updates MIIND and compiles it on the virtual machine. However, as MIIND is stored on the virtual machine, it can only be accessed via the cloud test bed and as mentioned, developer might not always have access to the cloud test bed, furthermore, the process of updating MIIND itself is quite time consuming. So, a solution to these problem is an auto-updater for MIIND which is stored on the cloud test bed itself, removing the issue with authorisation and connection to the cloud test-bed. The developer uploads the latest MIIND version to cvs repository (Concurrent Versions System, a version control system that allow recording history of source files), the auto-updater is automatically executed by a schedule and checks for the latest version of MIIND in the cvs repository, it then updates MIIND on the virtual machine.

4.3.2 Implementation approach

This task requires dealing with bash-script writing and scheduling. The issue lies with how and when to make an update. For example, the program needs to decide which version is working, which is up-to-date. There are several ways of doing this, a tag can be included to identify the version or any new changes, the alternative is to use CVS command and check the version but considering the limit interaction between the cloud test bed and the outside world websites, the former approach is more suitable. The frequency of update checking can be scheduled to be daily.

4.3.3 Evaluation

This task is technical related mostly. Evaluation is carried out mainly on technical aspects. Does the updater work properly? Does it manage to update the new version? Can it decide correctly when to update. The testing is carried out by letting the program run and update MIIND, changes are made to MIIND program to test if the program can detect a new version.

4.4 Porting MIIND optimisation

4.4.1 Description and brief solution

MIIND has a requirement for large amount of storage and computer resource. Using cloud to distribute the component has greatly increased the performance and utility. However the main component MIIND-Worker (the simulation processing) which takes up the most computer resource, the task of optimising MIIND is not yet completed. To confront this problem, parallelisation seem to be the best option.

4.4.2 Implementation approach

The most suitable design is distributing the simulation on each network node or a group of nodes to different VM as the design can be reused with Xen and OpenNebula, integrating the two design becomes less complex. This suggestion was also mentioned in original Porting MIIND report, section C.1 [18]

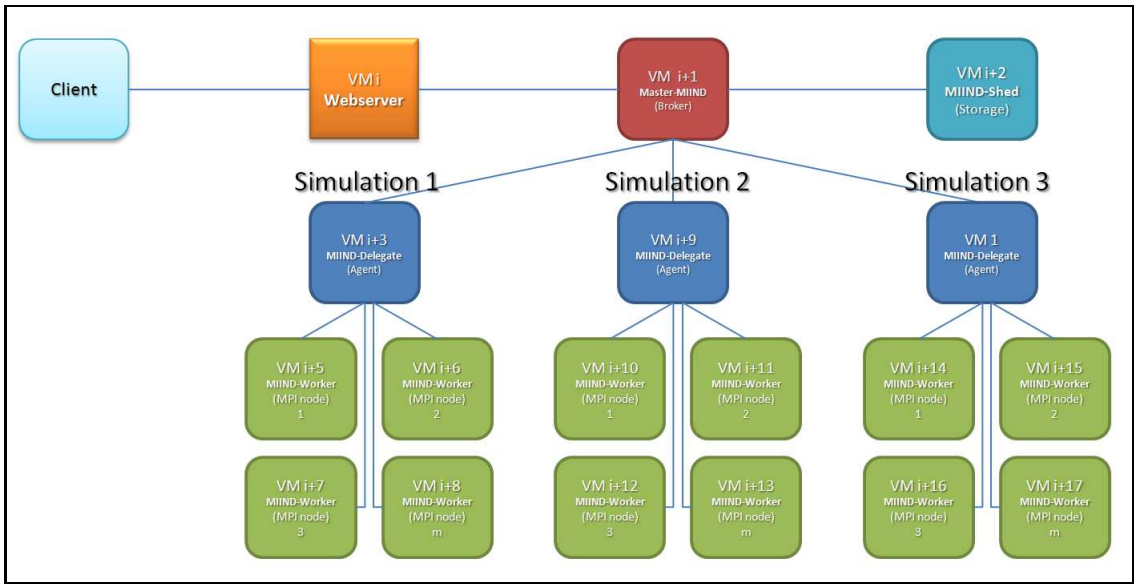


Figure 7: MIIND parallelisation [18]

4.4.3 Evaluation

The testing can be carried out with different input configurations and environment. Time taken in each case can be compared to confirm the stability of the application along with bug and error checking. The average parallel time is compared with serialisation time to draw the efficiency. Additionally, the test can be done with multiple users to observe the time and efficiency.

5 Porting MIIND extension

A more recent version of Porting MIIND was sent by Bjorn during week 8, which contains some additional components and functions. This version is different from the one described in original Porting MIIND report and it was the version of the Porting MIIND that I worked on. To keep the information consistent and avoid confusion when present my work, I will describe the whole work flow of Porting MIIND again in "XML usage implementation section" and go a bit more detail into the sections that I modified. There are sections which have already been explained in original Porting MIIND report; however, most of these were described from my own understanding while implementing the extensions.

My modification to Porting MIIND including:

- The XML webpage interface; add an option allows file uploading, create a XML parser to validate the file, modify the Linux-shell script so that it transfer the XML file instead of Workflow.plotpar and Workflow.runpar.
- XML file handler on MIIND-Worker; modify how the simulation is run using XML file and input and output of the visualisation process.
- The "Multiple XML request" implementation.
- The "WorkFarm" system.
- The "WorkRate" system.

The following sections describe about the implementation of the "XML usage" implementation (the first version where user can only upload one XML file), the "Multiple XML Request" implementation and the "WorkFarm System". One of the reasons for separating these sections is that I consider them to be separated versions of the Porting MIIND in term of design with different draw backs and issues, which will be explained more detailed in each section. Moreover, I also want to explain the implementation in relate with the progression of the project, which, in my opinion, would give the reader a better way to understand development stages of the project, furthermore, by separating these section, it is easier to specify clearly my parts of the implementation.

5.1 The XML usage implementation

The first XML implementation is designed very close to the original Porting MIND, there is almost no change to the Master-MIIND, and most of the modifications are built as external functions, which can be modified, removed or added at will without changing the design of the original Porting MIIND too much. This section explains in detail about how each of the components works in the XML implementation.

5.1.1 Master-MIIND

Master MIIND server is started by creating and running an instance of "MIINDServer.java". It first reads the Master-MIIND properties file "MIINDsetup.properties" and set the cloud network configuration values (masterIP, masterServer Port, WebServer IP, the virtual machine user-account, MIIND-Shed IP, the maximum amount of simulation allowed, the limit range IP that can be used, the number of virtual machine necessary) and the option that allows automatically starting instances other essential virtual machinies. If no properties file is found or the unreadable, the parameters are set by default. Unless the user wishes to handle the start-up manually, "MIINDServer" will initialise two "MIINDInstance"s starting Web-MIIND and MIIND-Shed virtual machines. "MIINDServer" then create an IP pool which generates and stores all the IP number (to assign to MIINDWorker instances) in the IP range. The IPs of the running machines are stored in a HashMap associate with the running "MIINDInstance", this is useful when administrator wants to shut down all the machines along with MasterMIIND server. The server then sets up a socket and waits for connection from Web-MIIND and MIIND-Workers. When a client connects, an instance of "MIINDMonitorClientHandler.java" is created to handle the client socket and run as a separated thread, the communication between client and server is handled by an instance of "Protocol.java". This protocol specifies client profile based on the received information and determine whether client is Web-MIIND or MIIND-Worker. If the client is Web-MIIND, the protocol creates an instance of "WebInteractor.java". The web interactor will first specify the request type "ACTIONTYPE". There are three request types.

- NEWSIMULATION requests a new simulation with new input XML file.
- RERUNANALYSIS requests visualising the simulation result. This part of code is disabled due to being incompatible with the new design.
- RETRIEVEARCHIVE requests the visualisation output images stored at MIIND-Shed to be sent back to Web-MIIND.

Only "NEWSIMULATION" request is explained in this report, the others are not part of my implementation and should be the same as in original Porting MIIND.

When Web-MIIND request a MIIND-Worker instance, web interactor locks an IP from the IPs pool, if there is no available IP, server sends an error message back to Web-MIIND, else the simulation ID is generated from the IP of MIIND-Worker (the first 14 digits are date and time when the MIIND-Worker instance is requested, the rest of the digits are IP of MIIND-Worker). The server sends a message containing the simulation ID (the simulation ID is also used as worker ID) and IP of MIIND-Worker virtual machine back to Web-MIIND. To prepare for receiving simulation input file, Master-MIIND sets up directories to store the simulation input and output data, after the XML file is transferred, an instance of "SimulationParticular.java" is created to assign the simulation parameters, prepares for a future MIIND-Worker, this instance is associated with the locked IP of the future worker in a HashMap. The server then checks if number of MIIND simulations exceeds the allowance, if not, an instance of "MIINDInstance.java" would be created to start up the virtual machine. When an instance of "MIINDInstance.java" is created, it will read in the OpenNebula properties file (opennebula.properties") and get the necessary parameters to interact with the OpenNebula API, in case the file is not found, default values will be used. After completes initialising parameters, "MIINDInstance.java" uses OpenNebulas API via RPC-XML to create a new virtual machine, in this case, a MIIND-Worker virtual machine will be instantiated. When connection between server and MIIND-Worker is established, same as the Web-MIIND, a new instance of "MIINDMonitorClientHandler.java" along with a new protocol are created to monitor MIIND-Worker. As described about how the protocol specifies the clients profile, to communicate with the MIIND-Worker, an instance of "WorkerInteractor.java" is created, updating the parameters with values from "SimulationParticulars" (user-name, simulation ID, action type and simulation parameters). With request type "NEWSIMULATION", the MIIND-Worker sends a message confirming that it is ready for processing, the server then sends back the simulation configuration values (simulation ID, user-name, output images format, archived result and action-type). After the client finishes initialising simulation configuration and setting up directories, the server will send the XML file to the client using Linux-shell script "dispatchConfig.sh" and request MIIND-Worker to start the simulation. After simulation is completed and output images is archived into a .tar file, server sends Web-MIIND IP to worker and request it to transfer the output archived file directly to Web-MIIND. If user requests the data to be archived and stored on MIIND-Shed, server sends MIIND-Shed IP to MIIND-Worker, after data is sent to MIIND-Shed, the server shut downs the worker.

However, there is no interaction between web interactor and worker interactor, thus, to inform Web-MIIND that simulation output archived file has been transferred; web interactor sets up a folder listener run in a separated thread. When MIIND-Worker transfers the simulation output file, it also sends a text file "complete.txt" which contains MIIND simulation execution times (the actual simulation time taken, the time taken to generate and convert images the time

when server finishes booting up MIIND-Worker virtual machine), when this text file is sent to the specific folder (.../result/watch), server will execute the Linux-shell script "dispatchProgressUpdate.sh" which transfers the "Workflow.log" to Web-MIIND along with a message to inform that the output files have been transferred. "MIINDServer" then shuts down the worker client handle and releases MIIND-Worker IP.

5.1.2 WebMIIND

The web interface now gives user the option to submit an XML file as input instead of typing in the input value.

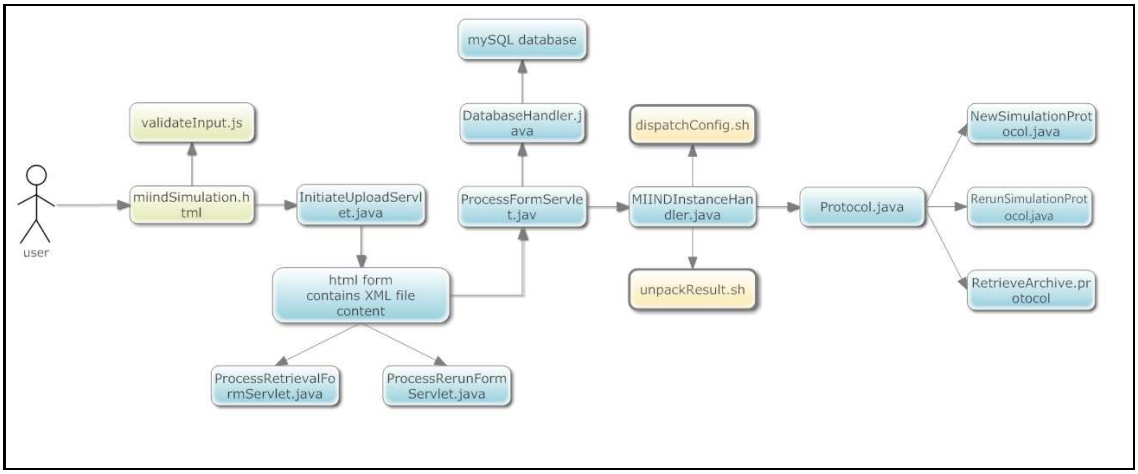


Figure 8: Web MIIND state diagram.

When user submits the XML files along with simulation preference setting (speed, storage and images conversion), an instance of "InitiateUploadServlet.java" is be created, To minimise the change made on the original code of Web-MIIND, "InitiateUploadServlet.java" is built as a version that specifies in dealing with XML file uploading of "InitiateRequestServlet.java" which is used in original Porting MIIND, both the XML usage version and the original are kept for testing purpose. The Java validation on server side validates the XML file name and size, the content of the file is also checked by parsing and validates the value of all parameters necessary. In case the parameters are missing, cannot be found or the file is unreadable, it will be treated as a corrupted file case and an html page will announce user. If all parameters are valid, the XML file content is copied into a String variable in Java when user submits the simulation input, "InitiateUploadServlet.java" then creates a hidden html form contain all the parameters including the content of the XML file, this page is auto-submitted and creates an instance of "ProcessFormServlet.java" which will read the parameters and the XML file content. "ProcessFormServlet.java" then connects to a SQL database used to store data on users and simulations, all interaction with the database is handled by "DatabaseHandler.java" including

accessing, querying, and updating, the required information to access the database is stored in "database.properties". To use the MIIND program, user needs an account which is stored in the database, an instance of "CommonServletFunctions.java" will validate login data of user, it also deals with writing the html content which displays the simulation status and ID to user.

To handle the interaction between Master-MIIND server and Web-MIIND client, "Process-FormServlet.java" creates an instance of "MIINDInstanceHandler.java" and transfer all simulation parameters values to it, "MIINDInstanceHandler.java" reads the cloud configuration from the file "webMIIND.properties" which contains the value of the server IP address, port and user-account used to execute Linux-shell script files (e.g. :scp command to transfer the file to server). It then tries to connect to Master-MIIND server, when connection is established, a specific protocol is set up depend on the request of user (Action type). In this case, which user requests for a new simulation run, an instance of "NewSimulationProtocol.java" is created to communicate with Master-MIIND server (if the request of user is Rerun analysis or retrieving archive, different protocol will be created to handle the communication). Upon initialising, the protocol sends a message to the server, informing of the client's type, user-name and simulation parameters (this message also defines the input type of the simulation: XML file or normal user-input-parameters). After receiving acknowledgement from server, Web-MIIND requests server to create and run a MIIND-Worker dedicated to the simulation. The server sends back the ID of the simulation, this ID is used to set up directories for storing simulation output data, the database updates the new ID associate with user-name. Web-MIIND informs the server of intending to send the XML file, after receiving confirmation, the string that contains the XML content is written to a file name "input.xml" stored in config folder and transferred to Master-MIIND using Linux-shell script "dispatchConfig.sh". After the file is transferred, Web-MIIND sends message asking server to proceed with the input file, the server then starts a MIIND-Worker virtual machine and sends Web-MIIND the time when starting time, this value is later used for measuring the amount of time taken to boot up MIIND-Worker. When simulation is finished, server sends output images and simulation log archived into a .tar file to Web-MIIND along with a message containing simulation execution time. A Java class "ServletVariables.java" is used to store the times value of the simulation (including: The actual simulation time, the images converting time and the time booting up MIIND-Worker). Web-MIIND then unpacks the .tar file and generates html document contains link to the images and log files; this process is carried out by Linux-shell script "unpackResults.sh". "MIINDInstanceHandler" then creates an html page to present the simulation result to user. The simulation log displays the simulation input parameters and the data of images output before and after conversion. For evaluation, MIIND execution times, cloud time (time taken for the whole request to be completed) and cloud overhead time (the amount of time spent on tasks other than the simulation) are calculated and written into a string for displaying to user on the web page. Finally, the database

updates the simulation status and Web-MIIND displays the web page to user.

5.1.3 MIIND-Worker

The MIIND-Worker virtual machine is started remotely by Master-MIIND, upon starting up, the client Java program "MIINDWorkerClient.java" is initialised via Linux-shell script run by a cronjob, it first reads in the properties file ("MIINDworker.properties") and retrieves server address. The client then sets up a socket and tries to connect to server using the value of Master-MIIND IP, port number and virtual machine user-name from the properties file. When connection establishes, a protocol is created to handle communication and requests from server. MIIND-Worker then identifies itself to server, configures simulation parameters values received from server and sets up directories to store simulation input and output files. Upon receiving the request to run the simulation, "MIINDWorkerClient.java" executes the Linux-shell script "runMIIND.sh". The script copies the XML file to MIIND program folder and run the simulation with the XML file as parameter.

There are two version of "XML usage" implementation in processing with visualisation. In the first version of MIIND (the original which was used at the start of the project), the function that allows producing image output of multiple nodes was not implemented in "Analyser" program despite in the XML file template, the format gives user the option to input multiple network node names used as parameter for analysing and visualising. Later, a new version of MIIND was updated on the cvs repository to solve the problem; this led to an alternated version of the XML implementation. In the new version, user is given the option to visualising multiple nodes but without using "Workflow" and "Analyser" programs; instead, both the simulation and the visualisation are carried out by MIIND executor (the main program which runs the simulation). This was the MIIND version that I used at the end of my project. Both implementations were kept for testing and developing, furthermore, the "Rerun Simulation" request requires the program to be capable of executing the analysing and visualising processes separately from the actual simulation (which means the first version where "Workflow" and "Analyser" programs are separated from the MIIND executor).

In the first version, after the root file (contains simulation result) is created, it is moved to "Workflow" folder to produce output images in postscript format. First, a test folder is created to store the output, the "Workflow" program will serialise the simulation result using the root file as parameter and generates output files in the test folder. The "Analyser" is then executed, output images are stored in test folder. The "Analyser" input parameter is not specified by Linux command; instead the program searches inside the test folder for the input file generated by "Workflow" program.

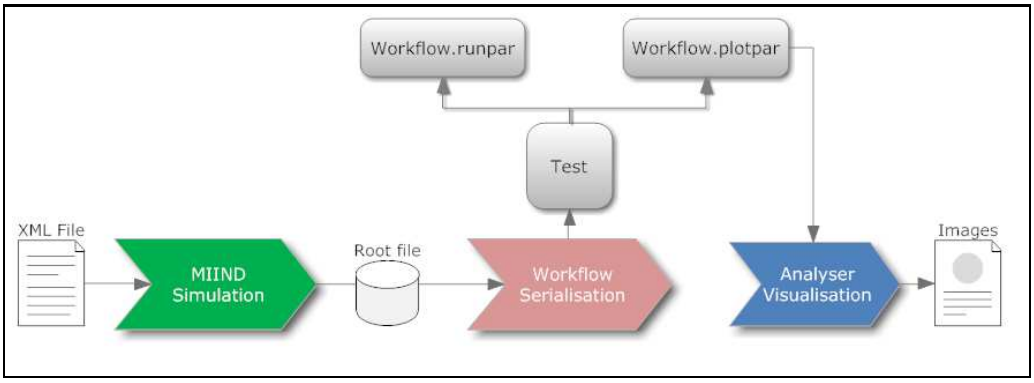


Figure 9: MIIND simulation processes.

The original Porting MIIND use two file as simulation inputs "Workflow.runpar" and "Workflow.plotpar". "Workflow.runpar" is the old simulation input which only contains several simulation configuration parameters; "Workflow.plotpar" contains setting values for visualisation. In the original plan, only the parameters included in "Workflow.runpar" were to be removed from the web interface, later on, Dr Marc DeKamps showed me that the XML file also contains the parameters for the visualising process (As explained in background research section, a MIIND simulation is run to simulate a network contains many individual nodes and the visualisation process is to visualise the activities of these node during the simulation. The information in "Workflow.plotpar" specifies which node to be visualised by their IDs). With this, we decided that "Workflow.plotpar" could also be removed from web interface. To take of advantage of this, the implementation now includes a program which extracts the information from XML file and builds a "Workflow.plotpar" file of its own. However, the nodes are specified by their name in the XML file instead of IDs (the ID is generated during simulation) as in "Workflow.plotpar". "Analyser" program only accepts node ID as input which would not work with directly using nodes names from the XML file. One of the methods to solve this problem was modifying the MIIND executor to be able to convert between nodes name and ID, this method was difficult as it involves modifying MIIND code that I was not familiar with and might change the program causing unknown behaviours. Another way was to modify the XML file structure so that it would use node ID instead of name, but this would also require modifying the XML generator program (which is also the MIIND executor), thus also has the drawback same as the first solution. Furthermore, it is convenient for user to choose node by name rather than ID as MIIND network nodes are usually labelled by names instead of IDs, using IDs might confuse users which node they want to be visualised. A simpler approach was to build a python or Java program to do the conversion between ID and name, the conversion is separated from MIIND simulator, making it easier for other developer to modify or remove. After considering all approaches, it was decided that a conversion built in MIIND-Worker client would be the most suitable solution.

Before “MIINDWorkerClient.java” executes “runMIIND.sh”, an instance of “Analyser.java” is created, it then parses the XML file and returns the simulation name (which is also the root file name, this is used for finding the correct root file to move to “Workflow” directory) and the name of the node that the user wants to be visualised.
An example of how a node is defined in the xml file:

```
<Node name="LIF E" type="EXCITATORY"
                    algorithm="Wilson-Cowan excitatory Algorithm">
```

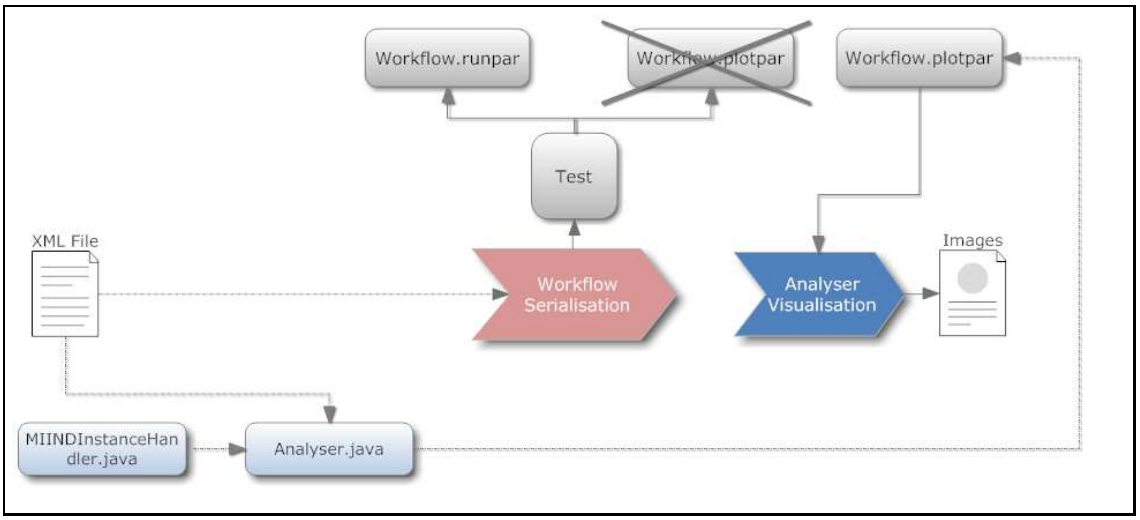


Figure 10: Visualisation input replacing.

Using the same mechanism of the MIIND simulation executor, “Analyser.java” assigns each node with an ID, it then creates a text file with the same structure as “Workflow.plotpar” and write in the IDs of nodes that requested to be visualised. The new “Workflow.plotpar” is copied to the test folder and overwrites the default “Workflow.plotpar” file right after the root file is serialised by Workflow program.
A default template of Workflow.plotpar:

```
<PlotParameter>
<NodeId>1</NodeId>
<TBegin>0</TBegin>
<TEnd>1</TEnd>
</PlotParameter>
```

In the second XML implementation, both the simulation and visualisation are carried out by the MIIND executor instead of using “Workflow”. The XML file is copied to MIIND executor directory and simulation is started using command “. /miind fig input.xml”. After the root file is created, MIIND executor serialises the result and visualises it, multiple nodes can now

be visualised and the output images are created in the test folder in postscript format. This implementation removes the need to converse between nodes IDs and their names from the XML file to execute the Analyser.

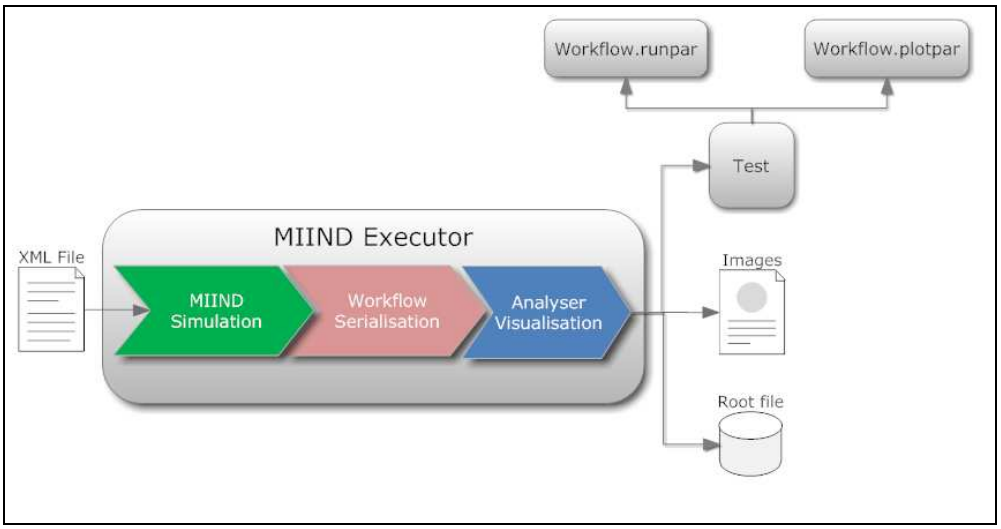


Figure 11: new MIIND simulation processes.

The output images contain the visualised node ID in their name, this was used for the original web-interface of Porting MIIND as the visualised node was specified by user using ID. However this might confused users which node to look at, now that nodes are specified by name in XML file. For convenience and user-friendly purpose, after the output images are created, "MIINDWorkerClient.java" will execute a Linux-shell script "RenameImage.sh" on all the images, replacing node ID by name.

```
The format of image name is:    grid_<Node ID>_<simulation time>.ps
The new format is:              grid_<Node name>_<simulation time>.ps
For example:
grid_1_0.050005811.ps    renamed to    grid_LIF E_0.050005811.ps
```

After the simulation is completed and images are generated, base on the "ImagesConversionType" value retrieved from the server, "MIINDWorkerClient.java" will execute the corresponded Linux-shell script and converse the images. MIIND-Worker then receives a message containing Web-MIIND IP and instruction to transfer the output files directly to Web-MIIND. "dispatchMIIND.sh" is executed; all output images are archived into a tar file and sent to Web-MIIND. A text file "complete.txt" is sent to Master-MIIND to inform that the result file has been dispatched, the Master-MIIND then informs Web-MIIND that the simulation is completed. To monitor the progress, another script file "dispatchProgressUpdate.sh" is run, copying "Workflow.log" file to Master-MIIND. To check whether user asks for result to be archived, MIIND-Worker sends a message to Master-MIIND, if results are requested to be sent

to MIIND-Shed, the server will send back MIIND-Shed IP address and MIIND-Worker runs the script "archiveData.sh" (depend on preference, output data might or might not be saved on the MIIND-Worker virtual machine) . The content inside the output folder ("test") is archived into a zip file and sent to MIIND-Shed stored in directories created remotely by Master-MIIND server. MIIND-Worker reports to the server that results have been archived; Master-MIIND will request the client to exit its program, shut down the MIIND-Worker virtual machine, remove it from the running virtual machine list, and release its IP. In case user does not request for archived result, MIIND-Worker will be requested to shut down instantly.

5.1.4 A look at the whole process

The following graph presents the whole communication processes happen during a request cycle of "XML implementation".

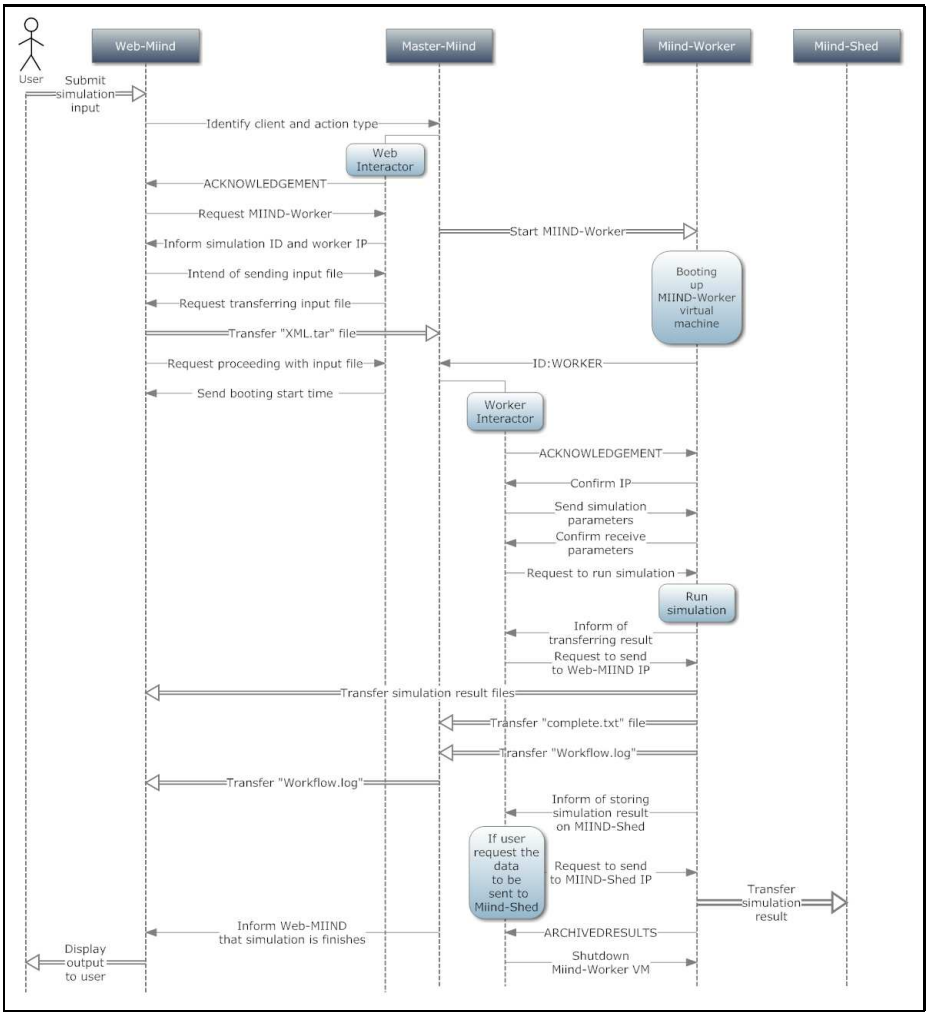


Figure 12: Communication sequence diagram.

5.2 Multiple XML Request

This can partly be seen as the updated version of the XML implementation as it is designed using the original as template, a large portion of code on both Web-MIIND and Master-MIIND are modified, changing the structure of many functions and even parts of the communication protocol between each virtual machine. There is little modification on MIIND-Worker.

5.2.1 Design and architecture

This extension was originally built for experimenting with simplifying the architecture of Porting MIIND. Instead of having one Web-MIIND instance for each simulation, the design is changed toward a one-to-many relationship between Web-MIIND and MIIND-Worker, saving extra communication for server.

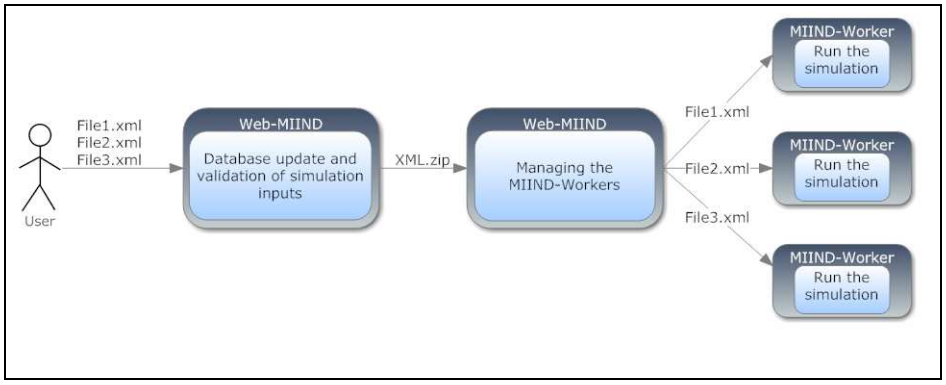


Figure 13: Multiple XML request design.

Later on, the objective changed to focus on user-friendly aspect, user can submit multiple simulation requests without typing in the parameters repeatedly.

5.2.2 Work flow

The html page now allows user to choose multiple XML files, it generates a files factory, reads all the XML files and stores them into a list. However, the old method of reading the XML content into a String variable causes issues if user submits more than 6 files, this is possibly due to the string content is too long and causes the program unable to process. The solution is to create a copy of each file; after validating the extension and simulation parameters, all XML contents are written back into their original files and stored in an unique folder; for each user, an unique ID will be generated to create a directory with the ID as the folder name (the ID contains the date combined with a random number, to avoid overlap, the program first check if a folder with the same ID already exists), the folder is a temporary location to store the XML files until

Web-MIIND receives the simulation ID and set up necessary directories, the directory path to this folder is written in the hidden html form. After "ProcessFormServlet.java" get the the XML directory path from the html form, "MIINDInstanceHandler.java" is created and sets up "NewSimulationProtocol.java" to communicate with Master-MIIND. When Web-MIIND sends a message to identify its profile, it also inform server of the type of simulation input (XML files or type-in parameters). In case the input type is XML, after receiving confirmation from server, Web-MIIND sends a message requesting the server to create MIIND-Worker running instance, the message also informs server of the number Worker required based on the number of XML files submitted (For each XML file, a Worker is started). Master-MIIND first check if there is enough available IPs, it then locks the number of Ips nescessary. The first IP is used to generate the main instance ID (this will be the ID to associate with Web-MIIND), other IPs are only used to generate ID by combining the first 14 digits of the main instance ID (date and time) with the IP number, Each of these IDs is treated as a separated simulation ID and later assigned to worker instance that handles the simulation. After a simulation ID is generated, it is assigned to the web interactor in a HashMap, this allows one-way interaction from worker interactor to Web-MIIND using its simulation ID. The server sends back to Web-MIIND a message containing the main instance ID and all the Worker IPs, Web-MIIND then updates the database and prepares for transferring the XML files to server. After directories are set up for each simulation ID, Linux-shell command is executed, copying all the XML files to the directory of the main instance ID, the files are then archived into a zip file and sent to Master-MIIND server along with a "Files.list" file which contains the name of all XML files. After sever receives the XML archived file, it creates a list of XML file names which worker can simply take the name from list and get the XML file. For each simulation ID, web interactor creates a folder listener to notify when the simulation result file has been sent to Web-MIIND, it then assigns each simulation Worker IP with a "SimulationParticular" storing all simulation configuration value and start booting up all the MIIND-Worker virtual machines.

To keep watch over all simulations, an instance of "WorkerWatcher.java" is created for monitoring each simulation status.

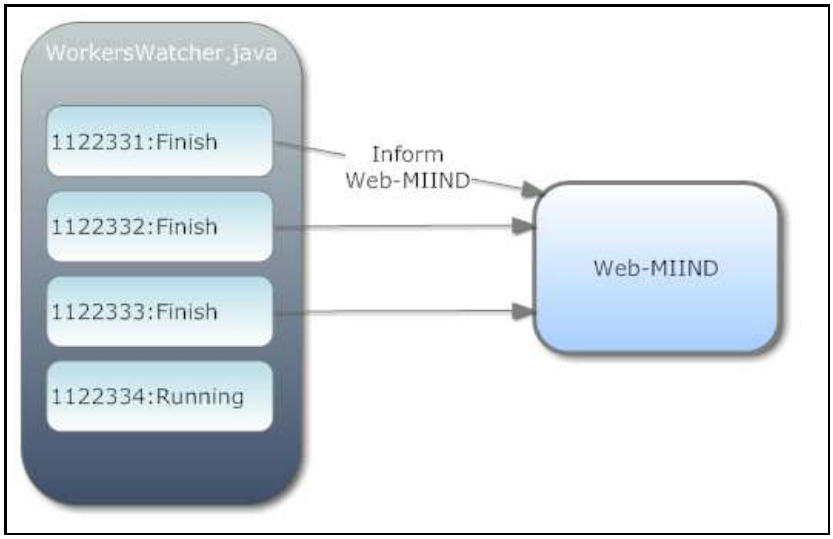


Figure 14: Workers watcher.

After connections are established with MIIND-Worker clients, worker interactor chooses an XML file name, it then calls the web interactor to sends the XML file name back to Web-MIIND to inform which XML file is assigned to which simulation. The name is removed from XML files names list and the file is then transferred to MIIND-Worker. After the simulation is completed, MIIND-Worker transfers the result file to Web-MIIND (the result file is stored in the directory identified by the simulation ID) and the text file "complete.txt" is sent to server. When the folder listener detects the file has been sent, "WorkerWatcher.java" will change the simulation status and send Web-MIIND a notification along with the MIIND execution time, the finished simulation ID and the "Workflow.log" file. To store simulation execution times and properties, "ServletVariables.java" creates a HashMap which associates the simulation ID with an object of "InstanceTime" which contains simulation execution times and other properties.

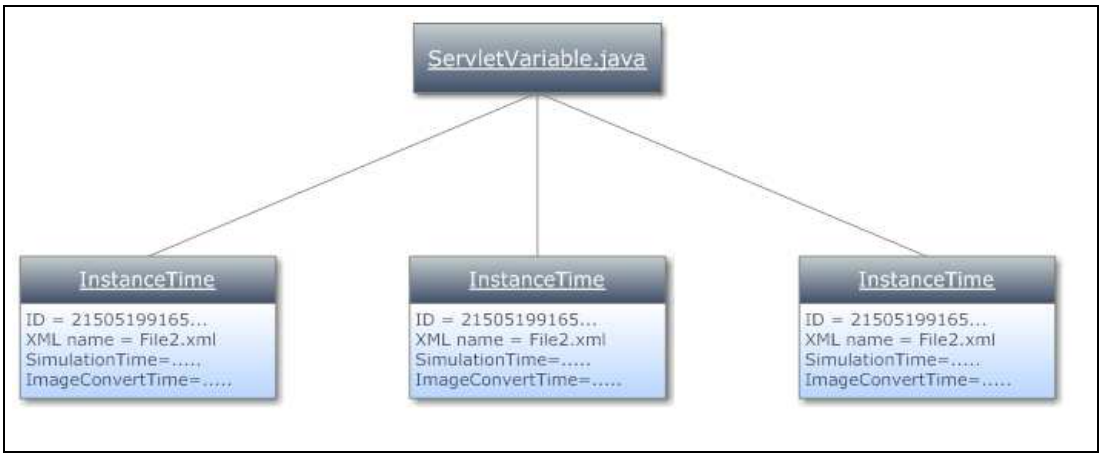


Figure 15: ServletVariable.java.

When all requested simulations statuses are set to finished, Master-MIIND informs Web-MIIND that all simulations result files have been transferred, the Web-MIIND handler is then terminated. Multiple simulation results are displayed on the webpage.

5.3 WorkFarm System

This extension was built mainly to solve the problem with MIIND-Worker virtual machine booting time, the booting times of virtual machines on OpenNebula level were very long, the amount of time spent on starting up and shutting down the MIIND-Worker contributed the most to the total execution time. After the completion of "Multiple XML Request", this system purpose also includes covering a major flaw in the design. When user submits a number of XML files, the server requests the same number MIIND-Worker instances at the same time, if there is not enough available workers, the server refuses user's request. This makes the program much less flexible in dealing with requests especially when it is close to limited number of Workers, user has to wait until there are enough MIIND-Worker instances available to send a request or reduce the number of XML files submitted. Moreover, the simulation request is strictly bound to its worker, if the MIIND-Worker virtual machine somehow fails to start up, the request is forever unfinished. Although it has the potential to address the above issues in "Multiple XML Request", this extension is considered a separated version of Porting MIIND design as its new architecture also introduces new drawbacks which are explained in evaluation section of this report.

5.3.1 Design and architecture

The problems of "Multiple XML Request" design mentioned are all in fact caused by the one-to-one design of original Porting MIIND which dedicates one MIIND-Worker to a certain simulation request. Thus, the purpose is to modify the design and makes server handling requests and workers more flexible. The most appropriate solution at time was to treats all simulation requests as a list of jobs in queue (new jobs are added at the end of the queue). Instead of dedicating one worker to a certain job, any worker can simply choose the first job in the queue (this is to ensure the "First come first serve" rule which enable requests being handled according to their submitting order), the job is then removed from the queue and next job is assigned to any worker available.

To tackle the issue with MIIND-Worker booting time, the currently only reasonable solution is to keep the worker running after simulation finishes, when user submits a new request, the server can simply assign a running worker to the request. This solution requires changing part of the communication process between Master-MIIND and MIIND-Worker, the simulation ID

which binds worker to a specific request also needs to be either removed or changed to a dynamic ID (changes when new request is processed on the worker).
Combining both of these designs, the final objective is to build a new architecture which allow workers flexibly choose their own job. After some research in work load distributing, I arrived to a solution called "Workfarm" system, which is a reminiscent of the work farm technique often used in parallelisation. [3].

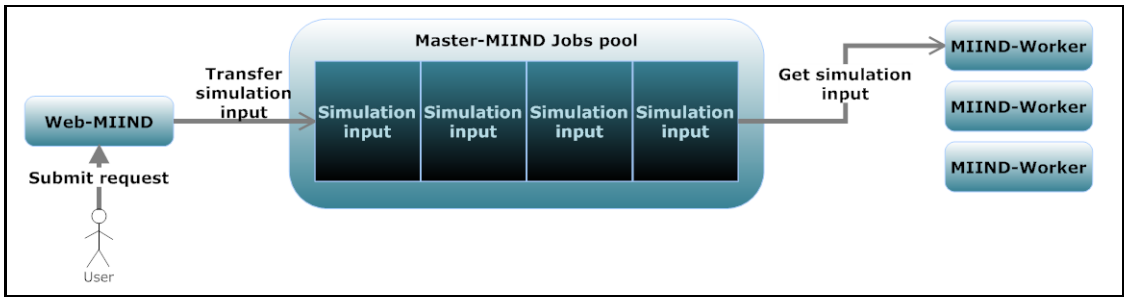


Figure 16: WorkFarm System Design

Requests submitted are put in a jobs pool. MIIND-Worker is started up independently (either booted up when server starts or when there is a need for more workers). When a worker finishes booting, it is put into stand-by state and search for a simulation request inside the jobs pool. After it finds a request, carries out and finishes the simulation, the worker is set to stand by state again.

Same as "Multiple XML Request", this implementation changes the whole original Porting MIIND into a new design with more loosely relationship between Web-MIIND and MIIND-Worker. Web-MIIND now completely has no information about the worker, worker can connect to server any time without being restricted to the user side actions (before, MIIND-Worker can only connect when there is a request from user). User can also submits any amount of requests without the fear of server being overload and requests not being processed with. This architecture guarantees that any requests will be carried out regardless of number of workers available.

5.3.2 Implementation

The Master-MIIND server now creates two pools, a workers pool and a jobs pool: The jobs pool is a list containing instances of "SimulationParticular.java" which is slightly modified to suit the new system. When user submits a XML file or simulation parameters, instead of requesting for a worker IP to bind to the request, Web-MIIND first ask for an unique ID for job (If user submit multiple XML files, each file is treated as a job), the ID is generated from combining current time, date with a counter that counts the number of jobs server receives (simulation result

directories are now built base on this ID). The ID is guaranteed to be unique as the requesting ID method is synchronised between all processes. Web-MIIND only needs to transfer the XML file or the config files ("Workflow.plotpar and Workflow.runpar") to Master-MIIND and the job will be registered into job pool waiting for a MIIND-Worker.
The workers pool contains IPs of MIIND-Workers that are ready to handle requests. (A MIIND-Worker is booted up when server starts running). When a worker virtual machine connects to the server, after handling all the communication processes to identify the worker, its IP is put in a waiting list and its worker interactor keeps checking the jobs pool. When there are simulation requests in the jobs pool, worker interactor takes the first job from the pool, it then reads in the job parameters values ("SimulationParticulars.java") and sends a message containing these value to its corresponded MIIND-Worker virtual machine, the worker IP is removed from waiting list. After MIIND-Worker finishes the request and transfers result to Web-MIIND, instead of being requested to shut down, the server will command it to change to stand by state, the worker IP is released and put back to waiting list and keeps checking for request again. In order to achieve a good performance by balancing the number of workers and jobs, a system called "WorkRateSystem" was created to manage the number of workers.

5.3.3 WorkRate system

This section describes the mechanism that "WorkFarm" system uses to manage number of MIIND-Worker virtual machines; The number of workers required to handle all requests is based on a "WorkRateSystem", this system introduces a "WorkRate" value that will determine whether MIIND server should boot up more workers and how many. This value is calculated using a formula requiring several parameters: the statistic value of average workers booting time, average simulation request processing, the total of number of running workers and the amount of request in jobs pool. The two time statistic values were originally planned to be measured by human observation, however, this would make the value static and not reliable in testing on other environments which might involve changing many factors such as; the work load on the cloud (other users might also be using the resource of the cloud which reduce the speed and storage), internet connection (delay in message transferring also contribute but if all virtual machines are run in the same cloud, this might not be an issue) or the MIIND simulation (simulation input can affect the amount of time taken to complete the simulation). So the problem is that the statistic values are not dynamic, thus, a solution is to update the "WorkRate" value automatically and frequently based on the program performance. As Porting MIIND handles more requests and more virtual machines are booted up, its performance will be changing as well, so to make sure that the system can accommodate with Porting MIIND, performance statistic values will be updated each time a virtual machine finishes booting or a request is completed.

The virtual machine booting times and request processing time are put in two lists. Whenever a "MIINDInstance.java" is initialised to start a MIIND-Worker, it records the time when the virtual machine is booted up, after the machine finishes booting, the program updates the booting time to the list. The same method applies to request processing time which is measured from when the worker receives a job until it finishes. To keep "WorkRate" value related to current performance of Porting MIIND, each data list has a limit number of values it can store (If the number of value in the list reaches the limit, instead of adding more, the first value which is also the oldest value is removed and new value is added at the end of the list). The maximum number of values in list is defined in a properties file "WorkRateSystemProp.properties", in case the file is not found, values are set to default. After the first request finishes (data has been updated to the list), whenever a worker receives a job, it will execute "WorkRateSystem" which first check if there are available workers waiting for job, if not, it then calculates the "WorkRate" value based on the formula:

$$WorkRate = \frac{((r \times (R + w))/w)}{b} \tag{1}$$

- b: The average amount of time to boot up one MIIND-Worker virtual machine.
- r: The average amount of time to finish one request.
- w: Number of workers running.
- R: Number of simulation requests in jobs pool.

The formula was built with the idea of using time to define the number of MIIND-Worker necessary. Main steps of the formula:

- Multiply the average time to finish one request with sum of running worker and current requests in jobs pool to estimate the total amount of time taken to finish all.
- Dividing the calculated value by the number of current running workers for individual time.
- Dividing the last calculated value by the average time to boot up one MIIND-Worker to estimate how many more workers would be enough to proceed with the available requests.

5.3.4 Server service change function

An extension function "Service change" was built, this function allows the service administrator to freely switch between the "WorkFarm" system and the "On Demand" system (which is similar to original one-to-one design of Porting MIIND). The reasons for this implementation are related to "Workfarm" system drawbacks which are included in evaluation section 5.

A file-change listener is now created in "MIINDServer.java" which keeps watch on a service status text file "MASTERMIINDHOME/service/servicestatus.sta". The file contains data about current service type of the server, when the file content is modified, the listener will inform server and change its service type (if the file is not found or cannot be read, service type will be changed to "On Demand"). With this design, we can manually control the server service by simply rewriting the service status file. When the service type is changed from "WorkFarm" to "On Demand", all current waiting workers will be shut down, any workers that have jobs will also be requested to shut down after they finish, for each requests submitted to server, a worker is booted up. If the service type is changed back to "WorkFarm", workers that finish their job will be put in stand by state and wait for requests. If there is no worker available and service type is "WorkFarm", a MIIND-Worker is started.

5.4 MIIND auto-updater

The purpose of the updater is daily checking and updating MIIND without requiring user interaction.

5.4.1 Architecture and Design

There are three machines required for the updater to be started: the cloud test bed, where the updater program is run, the MIIND-Worker virtual machine which stores MIIND program and a machine which can interacts with the cvs repository website.

As the cloud test-bed belongs to the school-own-private cloud, interaction with the outside world websites is limited, this prevents direct execution of downloading or updating (cvs checkout) MIIND on either the MIIND-Worker virtual machine or the cloud test bed. The solution is to update MIIND indirectly, a computer which allows interaction with MIIND repository is necessary to act as a "Router". During the project, I used "cslin-gps" machine of the school as the "Router" (the machine can be accessed by many different user account which avoids authority issue and the storage is large enough, these ensure that the updating process can be performed without problems), the updater program was stored on the cloud test-bed "testgrid3" and MIIND-Worker virtual machine was run at IP address 10.0.13.4. The "Router" and address of MIIND-Worker virtual machine can be changed by modifying the properties file "Properties.prop" which contains value of user account to access the "Router", the host name or IP of "Router", user account to access the MIIND-Worker virtual machine, the IP address of MIIND-Worker virtual machine and directories of the program. There are a few reasons why the cloud test-bed but not the virtual machine is used to store and run the program; direct data transfer from machines outside the cloud test-bed to MIIND-Worker virtual machine is not possible as the machine can only be accessed via the cloud, furthermore, it is also necessary to make sure the MIIND-Worker is running else it needs to be started up, the command to

start a virtual machine can only be performed on the cloud.

The MIIND auto update was built mostly using Linux-shell script and python. The program scripts are all stored on the cloud test-bed, to simplify the design and centralise the script files, I feel that this is appropriate as both the MIIND-Worker machine and the "Router" are not guaranteed to be always available, there are circumstances where the "Router" needs to be changed (for example: storage problem, authority or connection,), The same applies to MIIND-Worker, this is the sole purpose of the properties file, the location where the updating processes are performed can be changed at will. All the script are stored in same folder, the main script file (startUpdate.sh) manages other script files to performed their tasks and handles trivial commands. There are tasks which can only be performed locally, thus, instead of completely execute command remotely using ssh, some script files are transferred to other machines and started remotely using ssh, this also avoids overuse of executing command remotely as it might create an amount of overhead communication time (the amount of time taken to send command from the cloud test-bed can be varied depend on the connection quality). Log files which record output messages of the updating process are located in folder named after date (the program is executed daily, which avoid overwriting the log file).

The script file "updateMIIND.sh" which performs MIIND downloading and updating is created by the python program "ScriptGenerate.py". Depend on whether MIIND has already been installed or not, the content of the file will be different to carry out the suitable action. There is no specific reason for this design other than my own interest in experimenting with different programming techniques.

5.4.2 Workflow

Normally, a cronjob is set to run the program at 0:00 am daily. A python program "Properties-Read.py" reads the properties file and parses necessary parameters to access other machines (host-name or IP address, user-account to access and path of directories where the updating process takes place), the main program script "startUpdate.sh" is then started .

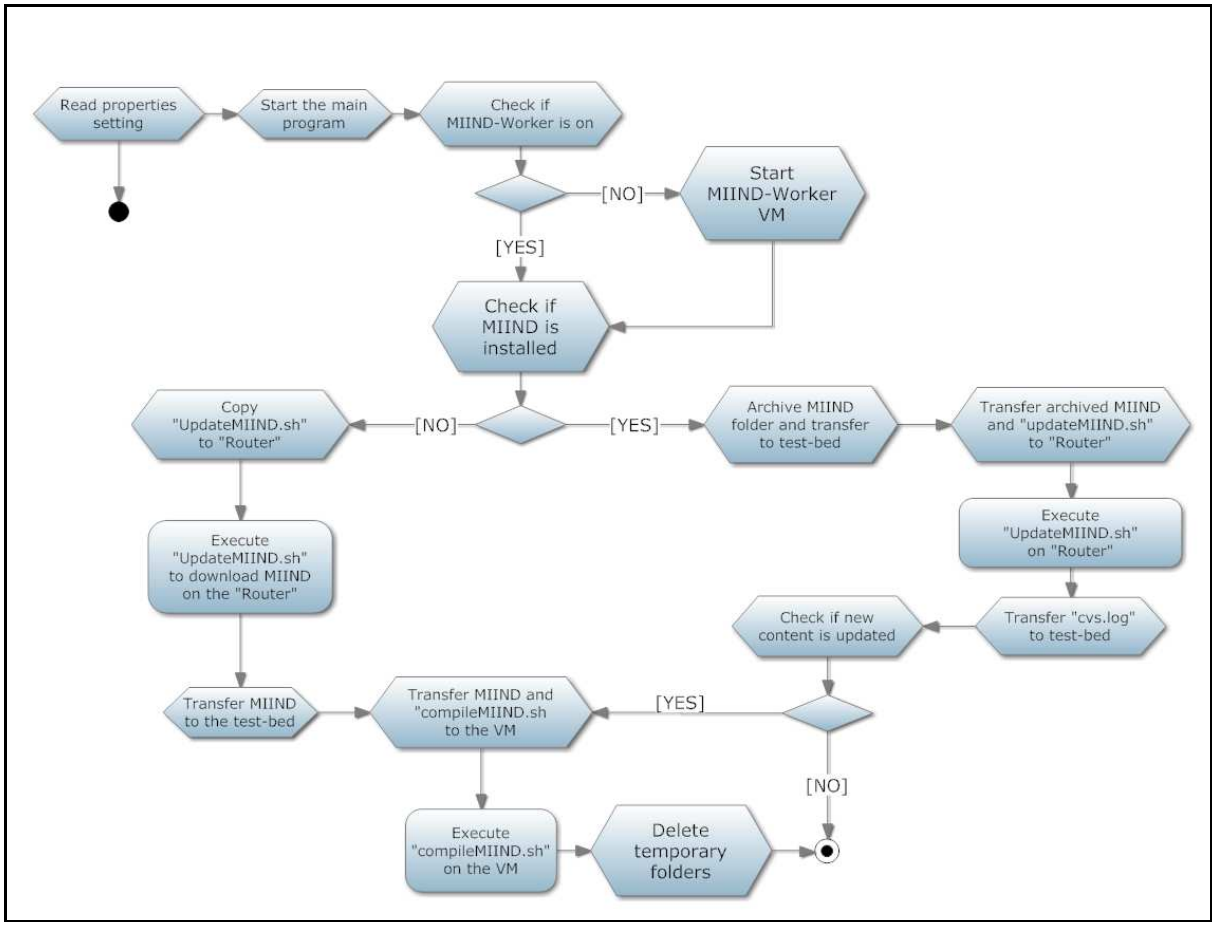


Figure 17: MIIND-updater work flow.

Using the parameters read from the properties file, the program creates a temporary directory (../MIIND-UPDATER) on each machine; MIIND-Worker virtual machine and the "Router", this directory is the location where the updating (or downloading and compiling) process is carried out, these directories are deleted after the updating is completed. To save time and resource, the program does not completely download a whole new MIIND program each time it updates, it first tries to access the MIIND-Worker virtual machine and checks whether MIIND has already been installed by simply locates the MIIND folder and see if it exists on the MIIND-Worker virtual machine (the folder is usually named "code").

In case MIIND has not been set up, the downloading script ("updateMIIND.sh") is transferred to "Router" and executed to download MIIND. After MIIND is downloaded, it is archived and transferred to MIIND-Worker virtual machine via the cloud test-bed. The program then copies the compiling script ("compileMIIND.sh") to MIIND-Worker and executes it. The archived MIIND file is moved to the installation directory and compiled. After the compilation finishes, the main program deletes the temporary directories on both MIIND-Worker virtual machine

and "Router", saves the log file and terminates.

If MIIND has already been installed, the MIIND program on the MIIND-Worker virtual machine is archived into a tar file and transferred to "Router" via the cloud test-bed. "updateMIIND.sh" is transferred to the temporary folder on "Router". It then executes the cvs update command, the output message on the shell terminal is saved into a log file (cvs.log), this log file is transferred to the cloud test-bed and read by a python script (checkMIIND.py) to check if there are any changes to the MIIND program. There are other methods of detecting new updated content; cvs diff command can be used to compare the working files with the revisions they were based on, and report any differences that are found, however this has to be performed on the machine where MIIND is compiled and as the interaction with outside world website is limited, this is not a solution. Another method is to compare the time stamp of all file in MIIND folder, this however might take more time to process, so instead of these two methods, Analysing the output log is a better and less complex approach to the solution, by simply reading every line in the log file, if there is at least one line informing of the change made to the files in MIIND folder, the updated MIIND program is archived and sent to MIIND-Worker virtual machine. The compiling script "compileMIIND.sh" is copied to the virtual machine and executed. If no new content is updated to MIIND, the program skips to final step; deletes the temporary directories, saves the log file and terminates.

6 Evaluation and further works

This chapter aims to evaluate the project achievement through a set of evaluation criteria. Justify whether minimum requirements and objectives have been met and to what extent. As this project requirement is software deliver and there are a number of programs; for each implementation, there will be analysing and explanation on whether the main objective of the implementation has been achieved and what are the flaws, drawbacks or any possible issues. In addition to the main purpose of the program, the evaluation is also carried out on performance aspect of part of the programs. From these, we can evaluate the limitation and flaws of the implementation, draw out conclusion and look ahead into what can be further extended or improved.To give reader a good understand of the further work suggested, this section evaluates each implementation, discusses its drawbacks and point out a potential solution to the problem. Aside from what is presented in this section, other remained potential further works are described in original Porting MIIND report: [18].It is difficult to include a solid proof that the implementations are successful, thus, there will be several screenshot in Appendix C to show what has been done. In this chapter, only the function validation and overall performance of each design are discussed and analysed.

6.1 XML usage implementation and Multiple XML Request

During the early testing stage (most of the test were for debugging and error checking), the test runs are carried out on XEN level, this is much less time consuming as on OpenNebula level since the boot time of the virtual machines are much faster than OpenNebula images, furthermore, change on XEN level is permanent which is convenient for debugging. All virtual machines and its Java client programs must be started manually as OpenNebula commands are disabled, however the advantage of this is that I can monitor the work flow of the components easily, (). I made several duplication of MIIND-Worker and create virtual images on different IP address. (Three MIIND-Workers were used for testing, located at 10.0.13.4, 10.0.14.5 and 10.0.14.6). Unfortunately, this testing method prevented me from foreseeing or understanding the possible issues on OpenNebula level. The two main problems were the long booting time of MIIND-Worker and virtual machines failing to start up, these prevent me from completely evaluate the performance more detail. In an actual cloud, these problems are unlikely to happend so often, thus there is little mention on solutions.

6.1.1 Main objective

Implementing and delivering the XML usage implementation. This objective was achieved; user can now submit XML files as simulation input. With the implementation of Multiple XML Request, the program allows starting many simulations simultaneously by submitting the request only once. Only one single Web-MIIND client is run, saving communication time. Testing was carried out together with the original XML implementation as their function is essential the same, moreover I also wanted to compare the performance between two design.

6.1.2 Drawbacks and present problems

The Rerun simulation request using XML file was obsolete, however the XML file handler implementation on MIIND-Worker also possesses the scope to carry out simulation result visualisation separated from the actual simulation, the java program Analyser.java which create the Workflow.plotpar can be run without executing the simulation when user only want to visualise the simulation result. Since the new version of MIIND does not remove Workflow and Analyser programs but leave them as an alternative option to run MIIND. This task should not be complex, although the option to visualise multiple network nodes would require discussion with Dr Marc Dekamps on development of MIIND.

Another issue lies with the images type, PDF format usually causes issues when viewed on the web-page and takes a lot of time to load, this is more of a third party software problem. JPEG is better for viewing directly on web-page if we do not take its conversion time into account, as its size is small and does not require running browser plug-in.

6.2 WorkFarm System

6.2.1 Main objective

With the implementation of WorkFarm system, MIIND-Worker long booting time issue was avoided to some extent; with workers machines continues running after they finish requests and the flexible system of jobs pool (which allow any job to be assigned to any worker), and users now mostly do not have to wait for the virtual machines to be booted up every time they submit simulation request. In case, there are not enough workers to handle all requests in reasonable time, WorkRate system will manage and increase the number of workers suitable as this system is based on measured performance of the current running server. According to Bjorn, unlike the school private cloud, the Amazone cloud was faster in booting virtual machine on OpenNeubla level. Thus, although, this implementation was an useful addition to Porting MIIND, greatly support testing and boost the performance of the program when used on the school private

cloud, the same cannot be said for all other cloud system but certainly, this extension will improve the overall performance to some extent. Although in original Porting MIIND report, this problem was mentioned, I underestimated its scale and did not understand the burden it creates on the overall performance, due to this, WorkFarm system was implemented during the testing stage of XML implementation instead of originally being planned.
WorkFarm also gives Porting MIIND a more flexible design, removing the strict binding between workers and simulation requests, allows requests to be handled in First come first serve rule regardless of the workers. This further extent the Multiple XML Requests implementation, user can submit any amount of requests as long as it does not exceed the limit number of simulations allowed simultaneously, all guaranteed to be carried with the condition that there is at least one worker running, this covers the problem with requests being unprocessed due to MIIND-Worker failing to start.

6.2.2 Performance test

A function which records the time taken to process with one request was implemented. The time is measured from when WorkerInteractor.java receives a job until it finishes. The data is saved in a data file RequestTimeStatistic.data, the file contain all information about a job (except user-name) and the time taken to process it. Only RetreiveArchive request is not measured as I considered the time taken is not significant enough to affect the program performance. I built this function to monitor the performance of worker after carrying out continuous request and see if the performance is worse or the same as when the worker was freshly started.

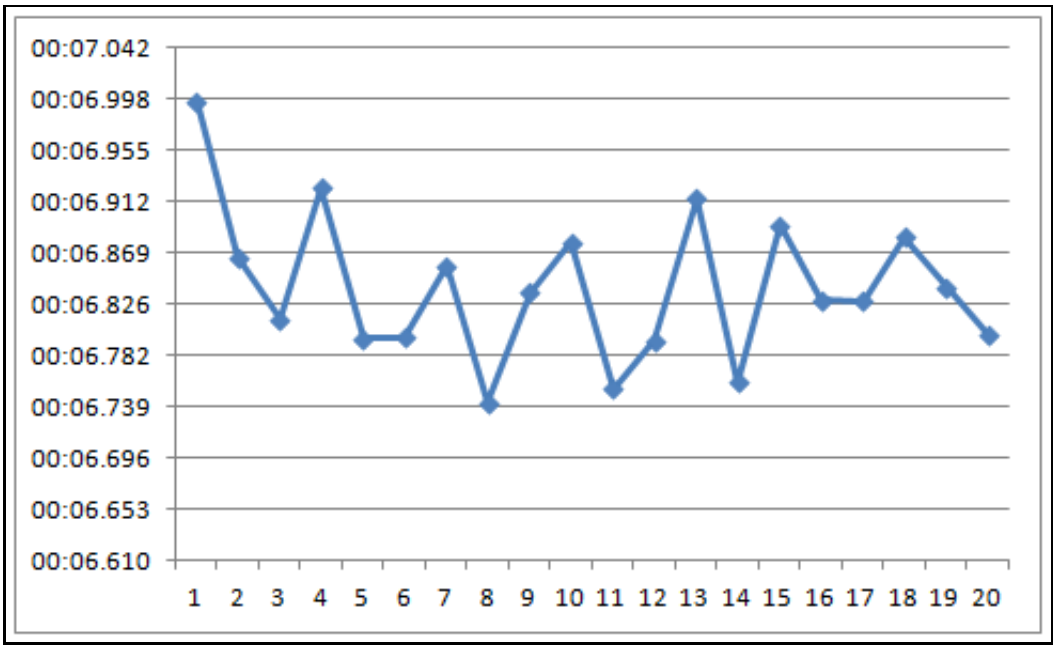


Figure 18: Requests processing time chart

Several requests were submit simultaneously to test the stability and performance of MIIND-Worker after dealing with large number of request, the result showed no issues, all requests were completed and displayed on web page. Performance speed of MIIND-Worker slightly changed on each request but no sign of constant decreasing.

For the purpose of monitoring OpenNebula performance, I also built a function using that executed along with WorkRate system to monitor the virtual machine booting time, it calculates the times and write them into a data file BootTimeStatistic.data, the file contents each virtual machine's machine type and its booting time. This function might also interest administrator of the cloud or anyone who wishes to study the cloud performance.

```
master@debian01 /usr/apps/MIIND-SaaS/Master-MIIND$ cat logs/BootTimeStatistic.data
VMTYPE:MIIND-Worker;BOOTTIME:5 minutes, 32 seconds, 194 milliseconds.
VMTYPE:MIIND-Worker;BOOTTIME:5 minutes, 46 seconds, 23 milliseconds.
VMTYPE:MIIND-Worker;BOOTTIME:6 minutes, 3 seconds, 728 milliseconds.
VMTYPE:MIIND-Worker;BOOTTIME:6 minutes, 7 seconds, 262 milliseconds.
VMTYPE:MIIND-Worker;BOOTTIME:6 minutes, 24 seconds, 899 milliseconds.
VMTYPE:MIIND-Worker;BOOTTIME:6 minutes, 34 seconds, 632 milliseconds.
```

Figure 19: Booting time and Request time monitoring

We can see that the amount of time taken to boot one worker is 5 minutes while the time to process one simulation request is roughly 6.8 second. In term of performance, "WorkFarm" system does indeed produce a much better result than the original Porting MIIND design.

6.2.3 Drawbacks and present problems

Architecture is actually one of the main drawback of this implementation, it removes the original idea of allocating user resource in Porting MIIND where each user has an isolated pool of resource dedicated to his/her resource. This design is completely re-modified in WorkFarm system; the server does not concern with which request is performed by which worker, it only concern with the simulation being processed with and result being transferred to user. Furthermore, the original Porting MIIND design takes cloud computing features into consideration as its priority: reduction in permanent IT resources and cost through "on demand, as many as necessary when necessary", take advantage of the elasticity the cloud offers and in an actual cloud where user has to pay for the amount of resource used (amount of time running machine and storage), the process of starting up and shutdown MIIND-Worker after finishing one simulation indeed supports the idea of pay as you use. However, WorkFarm design keeps the worker running after simulation finished, this would produce a lot of wasted resource if there are only a few simulation requests submitted simultaneously and the cost would not be worth the amount of time booting up MIIND-Worker. But considering in performance aspect

when a large amount of requests being submitted continuously to server, we cannot deny the overwhelming advantages WorkFarm would introduce.

A temporary solution was implemented, a function which allow user to change the service type between WorkFarm and Ondemand depend on the amount of requests at time (at peak demand time, server can change to WorkFarm and the , this was described in chapter 2 Design and implementation. So far, the function is still manual operated by cloud user, but as it is built with flexibility and intention for future development, when an application which can automatically decide between WorkFarm and Ondemand is implemented; it can be easily merged with the service change function with little effort. One simple suggestions on how the service change can be automated is to observe the amount of requests according to time daily (For example, from 9 am to 3 pm, there are a larger number of requests than other times), then create a cronjob to handle the change. This was not implemented as I do not have an actual testing environment (lack in number of users as well as an actual cloud).

Worker settings, WorkFarm system also removes the option of choosing speed priority and memory storage. This feature was implemented to give the user some flexibility as to "priority", eg a simulation with lots of data processing might be faster if machine specs are increased (such as memory/CPU), but with pre-booted workers, this is difficult to achieve as all options have to be defined before booting the virtual machine unless there is a way to change an already running virtual machine settings.

A possible solution would be pre-booting MIIND-Worker machines of varying specs and put in waiting pool when the server is set up, distributing the number of MIIND-Worker virtual machine equally into three groups: the High priority group, the Medium priority group and the Medium priority group, when Web-MIIND requests for an MIIND-Worker virtual machine, the server will choose a worker. If user does not specify the speed priority of the virtual machine, a default mechanism would set the priority to Medium, the same method can be applied to memory storage option, but this would create a lot of Workers running which could affect storage and performance. To make sure the right worker is assigned to a suitable task, high spec machines would be kept at the end of the pool and assigned by preference to proceed with simulation requests that required a high speed priority. In case there is no MIIND-Worker virtual machine that satisfies the preference setting, other machine would take the job instead, to ensure that computer resource is managed well, only machine that has speed priority setting closed to the preference will be used (If the preference is high speed priority and there is no high priority worker, medium priority workers would be used and in case there is no low priority one, medium priority workers are also used).

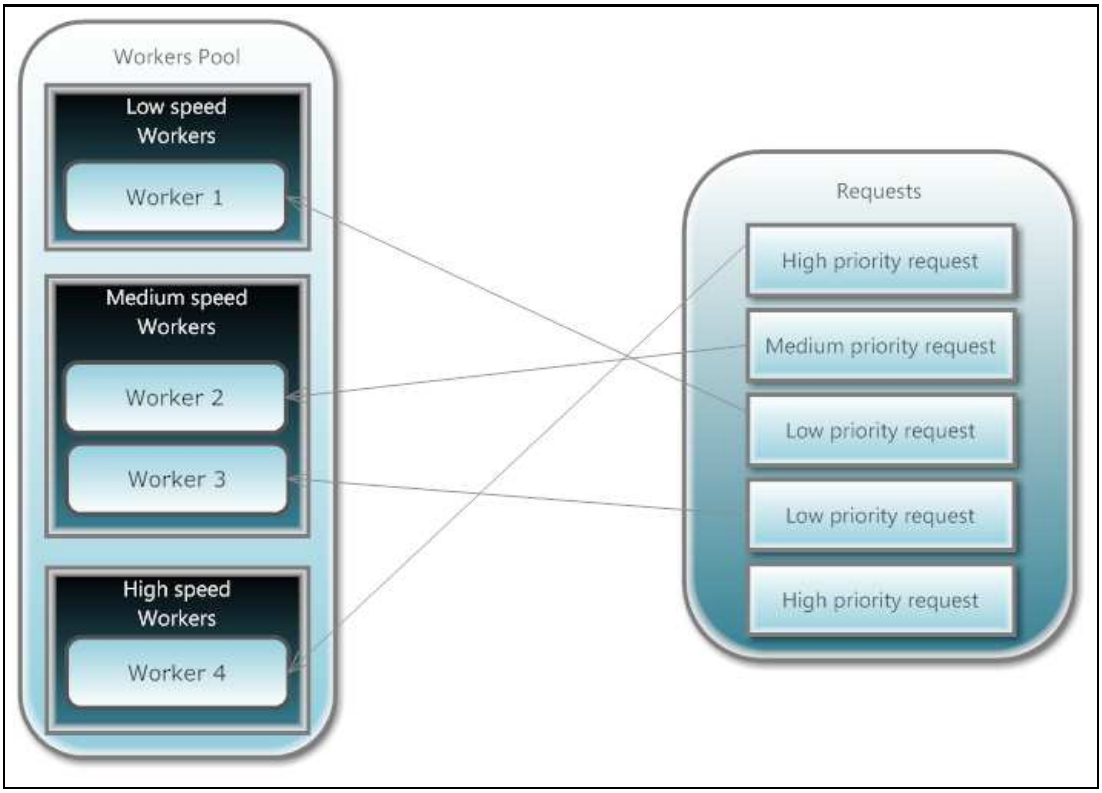


Figure 20: Workers distribution

This can prevent the lack of computer resource issue with high priority job and reduce the amount of wasted resource when there is not enough low priority workers. With this solution, we could still present the option to choose speed priority to user, but do not guarantee that every request of user would be processed by MIIND-Worker with the preference setting, this somewhat move the design toward grid architecture as ideally, in the cloud, we would be able execute at the priority indicated as the resources required should be available and can be accessed any time, but we're both limited by the test-bed, the current flexibility of our software solution and, on a public cloud, how much we or the user are willing to pay for computer resource and storage. As the Worker virtual machine being kept for a long time, storage and computer resource is not the same as a new fresh-booted Worker, this could lead to some performance reducing. The virtual machines can be shut down once daily to save storage and computer resource. Also, when Web-MIIND requests for a MIIND-Worker,

Another slightly different design to present the speed priority and storage feature is to assign request only to worker that has the preference setting. This can be done by strictly separating workers into three pool (a high speed priority pool, a medium pool and a low pool). If there is no worker in the requested pool, the server will start a new worker with the preference setting. This method guarantees that users will always have the worker they prefer and requests can be proceeded without the risk of crashing or problem relate to computer resource. This implemen-

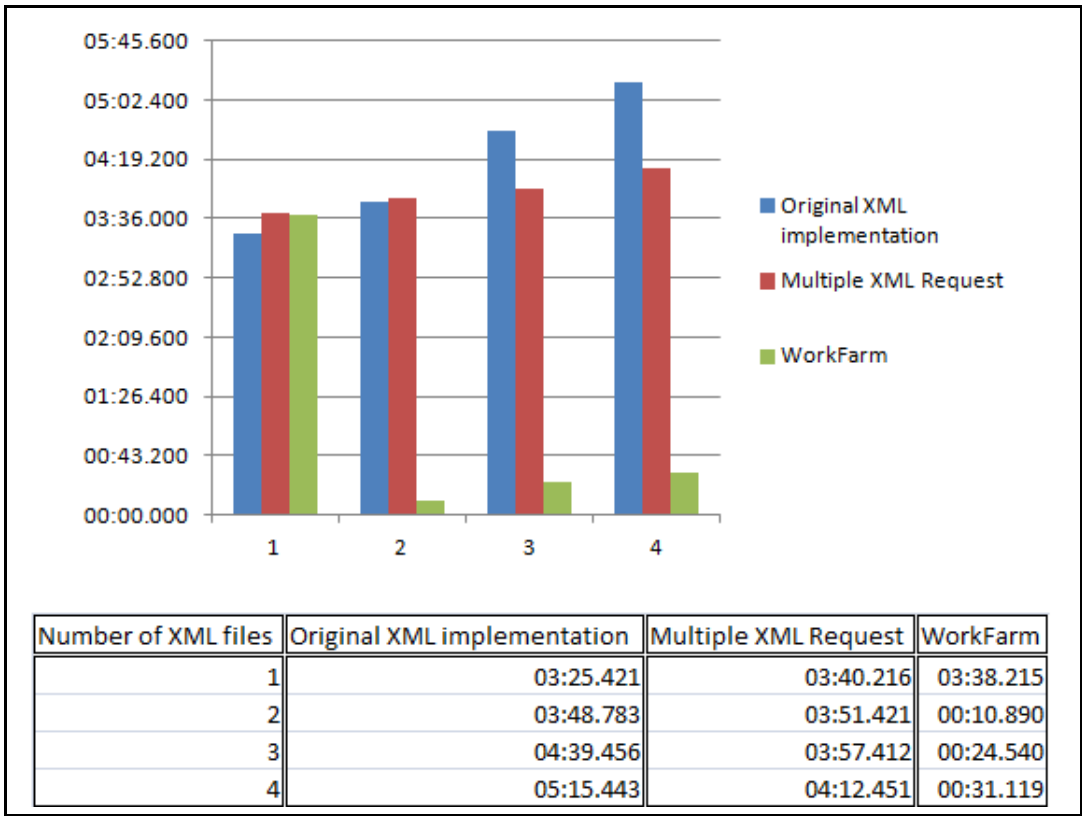
tation was considered in the plan, however, the drawback is that unused workers would become wasted resource and storage which might slow down the cloud performance significantly (The server might not be able to find any MIINDInstance that has compatible set up with user request despite there are quite a lot of MIIND-Workers waiting for request. This might reduce the performance of the computer where MIIND-Worker virtual machines are created significantly, in worst case, the number of MIIND-Worker might even excess the allowed limit while there are still a lot of MIIND-Workers on stand by)Not to mention if we want to do the same for storage option, the amount of resource used to boot up all worker would not be feasible. Still, this can be solved by shutting down other Workers to claim resource and storage for the new one, but it would certainly require user to wait longer for the virtual machine to be shut down and backfire the objective of increasing performance and cut down on virtual machine booting time.

There is a possible problem with the "WorkRate" system, in actual usage of MIIND, simulations are various, some are more intensive than other, managing the number of workers based on average requests processing time might not be a good approach. Further work can be done to modify how the work rate value is calculated so that the system can manage the number of workers reasonablely. This will need testing with various XML files producing different simulations.

6.3 Performance

To compare the performance of three implementation (the original Porting MIIND design, the Multiple XML Request and the WorkFarm), several tests run were carried out with different number of XML files submitted simultaneously. Speed priority and memory storage options were not included in the test system due to these feature are removed on WorkFarm system implementation. The number of requests submitted at same time was limit at 4 as when with 5 or more requests submitted simultaneously, the cloud randomly failed to boot up Worker virtual machine. This is also one of the reasons why speed priority was not included in the test; with High speed priority, the virtual machine seemed more likely to fail during booting process. The cause of this could have been due to limit computer resource and storage as there were other users using the cloud as well. It was not easy to have a chance when I can completely use all the cloud resource for testing since most students proceeded with testing and evaluations during the end month of the project. The problem also might have been my implemented design, though the chance of this is unlikely as the virtual machine booting process is carried out by OpenNebula. The result of this problem was that it limited the number of Worker VM I could initiate to work with, thus prevented testing with a large number of simulations submitted simultaneously. Only the total cloud times (the total amount of time taken to complete request) was compared as I considers MIIND-Booting time, simulation time and images conversion time are not related to the implementations design. To ensure reliable data, all XML files have the

same content (same simulation input, same parameters).



Number of XML files	Original XML implementation	Multiple XML Request	WorkFarm
1	03:25.421	03:40.216	03:38.215
2	03:48.783	03:51.421	00:10.890
3	04:39.456	03:57.412	00:24.540
4	05:15.443	04:12.451	00:31.119

Figure 21: Comparing cloud Time

With only one XML file as input, the overall performance of the original design seem to be the best as other implementations features were not exposed. (WorkFarm system has to start a MIIND-Worker on the first request, thus its total time inlcudes the worker booting time and is similar to others.) With two XML files as input, WorkFarm system prove its overwhelming performance in handling multiple requests. Multiple XML Request is still slower than original XML implementation, however the gap between the two is close. With three and four XML files as input, we can clearly see the difference in performance of three designs. The amount of time difference between Multiple XML and the original proves that the communication process between Master-MIIND and Web-MIIND does contribute an amount to the total time.

6.4 MIIND-updater

6.4.1 Main objective

Implementing and deliver the MIIND auto-update application. Main objective of the implementation has been achieved; the MIIND program stored on virtual machine MIIND-Worker run at IP 10.0.13.4 is now checked for update daily. The program managed to detect new up-

date content when Marc updated MIIND on the cvs repository. With this, developer can now indirectly modify the MIIND program stored on MIIND-Worker virtual machine by updating the new version on cvs repository without the need for accessing to the cloud.

6.4.2 Drawbacks and present problems

To allow completely automatic updating, the cloud test-bed (testgrid3) was allowed accessing to router (cslin-gps) computer under user-name sc09mn and the MIIND-Worker virtual machine 10.0.13.4 using user-name root without the need for any password. This however is not a secured approach as any person accesses the cloud test-bed with sc09mn user-name can also freely access cslin-gps (although any users able to access the cloud test-bed should also have authority to cslin-gps, thus it would not be a serious problem); furthermore, an actual cloud would be able to configure the security setting which enables interaction with certain website and allows direct update or check out cvs command on the virtual machine. In this case, some modifications need to be applied; remove the authorisation key that allows accessing without password from the cloud test-bed to the "Router" (cslin-gps) and modify the script to perform updating directly on the virtual machine. As a properties file is presented for developer to change the user-name for accessing the cloud test-bed, I suggest using a more secured account instead of sc09mn. Another possible issue is when boot up MIIND-Worker virtual machine on XEN level, there might be other machine running on the same IP 10.0.13.4 (e.g: its OpenNebula images), this would prevent the worker from being started and the updater cannot proceed. The solution can simply be start MIIND-Worker on a unique IP address which would guarantee its being started up correctly. This might need some discussion with an expert on the cloud test-bed (Django ArmStrong).

6.5 Client Opinion

Dr Marc Dekamps was sastified with the result of XML usage implementation. He also noted that the "Multiple XML Request" and "WorkFarm" systems can be useful additions to the Porting MIIND program. There was not enough time to involve other researchers or students in testing user-friendly aspect.

7 Conclusion

As this project does not have a specific focused objectives, it is difficult to gauge the how successful it was, but from what have been achieved, the minimum requirements of the project were delivered and several additional objectives were also accomplished, the expectation certainly has been exceeded. Users who want to use Porting MIIND can now take advantages of the XML usage implementation, giving them its benefits. A MIIND-updater was also built, allowing developer to daily update the program without having to access the virtual machine

directly. Aside from the minimum requirement, a new architecture WorkFarm system is introduced, providing Porting MIIND a new design with new features and benefits but at the same time also removes some of the basic purpose of the original version. Although, there are both new improvements and drawbacks, this project can be considered to have further extended and improved Porting MIIND.

8 Appendices

8.1 Appendice A - Personal Reflection

A project which involves extending an existing work is usually quite different from developing your own, there are both challenges and benefits. The main obstacle with this type of project is studying the existing code and design; this may depend on the scale of work and how the implementation is documented. Porting MIIND architecture is quite straight forward with its 3-tier design, however, the actual code is a lot more complex as it involves dealing with various tasks and functions, I managed to understand the implementation well thanks to reasonable description in the project report. On the other hand, the benefit is that, a clear direction of the design has already been laid out; we can easily decide the plan and method based on this direction. For example: the 3-tier architecture helped me decide where I could focus my modification when I wanted to change a certain process or function of the program (XML file handle on MIIND-Worker, XML file validation on Web-MIIND, requests and workers management on Master-MIIND). Personally, I also find that by studying an existing program, my analysing and designing skill have improved much compared to the beginning of the project.

Porting MIIND involves several third-party software and tools. The main obstacle that prevent my progression the most was the requirement of knowledge and experience in interacting with Xen and OpenNebula, issues that sometimes are very simple could prevent progression of the whole project. (E.g. Setting up Porting MIIND took more time than expected, from week 6 until week 9, this was also the main factor delaying the project progression, I could not run the program on Xen level, the problem took about one week until I assumed it was because Lynx, a text-based browser which had some limit functions that caused Web-Miind unable to receive the input parameters from user. Thanks to Bjorns and Djangos assistance, the problem was solved by using Port forwarding from the local computer to the cloud test-bed). From this experience, my first realisation was that in every project concerning third-party software and tools, it is essential to be able to contact experts with specialities related to the software or research field. This also applies to MIIND which thanks to Dr Marc DeKamps, I was able to understand the program work flow to design a suitable extension compatible with the program. Obviously, as Porting MIIND is the main component of the project, it was extremely important

to keep in contact with its original developer, Bjorn Gerckens. There were many sections of the lower level code unmentioned in the report which required a deep understanding of the program in order to modify them. The conclusion is that in a project of this scale, the work involved does not just surround coding, gathering material and information in fact also plays an equally important role to make a successful project, objectives can hardly be achieved alone without the knowledge of all research fields involved.

An important aspect of the project, which I was not able to comprehend; it is essential to have frequent meeting with your supervisor, not only to discuss about the progression of the project but also to consider new objectives, possible extension and any changes occur. (The WorkFarm system could have been put as an actual extension had I discussed with Marc about the virtual machine boot time).

An unexpected obstacle was the Linux environment. I struggled a bit with working on Linux platform despite I had been using Linux during 3 years at the University, as the extent of Linux operations used in the project were higher level and related more to cloud computing and networking. However this was not a big challenge as documents on Linux-shell scripting can easily be found, unlike cutting-edge software like Xen and OpenNebula. By the end of the project, I found myself relied much more on Linux (especially the MIIND-updater implementation) as it provides easy approach to certain functions which can be executed by several simple lines of command instead of a complex set up in other programming language (the function to transfer file between machines can be performed with a single command of scp while in Java a number of steps is required such as: setting up socket, create a buffer reader, input file stream and converting the file from binary to object and object to file).

At the beginning of the project, I misjudged the scale of the writing task, it was certainly not easy. Usually, to avoid a large amount of work load piling up, it is reasonable to keep writing as progress, but the content might still need many modifications later. As the project progressed, new implementations are added and new problems, issues occurred which requires modifying a whole section (the late implementation of WorkFarm system was at first treated as possible future extension to the project.). And there is also the problem in using tense, by trying to assume being at the end of the project, I did manage to avoid changing the tense in most sections, but at the same time, this also created unnatural and confused paragraphs which at the end required a lot of rewriting. A lesson I learned at the end is that it is important to keep the objectives consistent and follow the schedule closely, this can avoid too much rewriting and a clear direction is established which would prevents sections from overlapping with each other.

For students who wish to involve with Cloud Computing in general and Porting MIIND specifi-

cally, it is most important to be able to contact with someone who has a complete understanding of the program, all the better if meeting in person can be arranged. To be able to modify the program, simply reading the report would not give the level of understanding necessary, studying the code and performing actual test runs will present the whole work flow and design much better. And lastly, do not be discouraged by the huddles of the project, with actual effort and assistance from experts in the field, there is enough time to adapt your knowledge to be capable of achieving the project objectives.

8.2 Appendice B - Resource used

8.2.1 The cloud test bed

Most of the actual coding and early test stage were carried out on the private cloud of The School of computing, specificly the cloud test-bed testgrid3 which allowed developing and testing freely without paying for the usage. Xen hypervisor and OpenNebula are used to manage and modify the virtual machine images.

8.3 Porting MIIND

The original Porting MIIND program was provided by Bjorn.

8.3.1 Programming language

The two main languages used are Java [1] and Linux-shell script, as they are also the main component of the original Porting MIIND. Furthermore, the environment is a private cloud which Java was the language that I had the most experience with in working with cloud architecture and network from CR21 and TC30 modules .

Linux-shell script was the most suitable for dealing with tasks that involved in interacting between files and directories on the Linux Operating System, the ssh protocol also allows various interaction between networked computers with less effort than in Java. In the auto-updater implementation, Linux-shell script is the main component as I tried to keep the program simple and centralised on one machine.

Python is also used, but mostly for trivial and general tasks due to its nature being a simple and easy to use language; no compilation required in testing and Python has a large amount of built-in library, each with various and useful function that can carry out common tasks with a very small amount of coding.

8.3.2 Libraries and tools

For the XML web-interface, Apache.common.Fileupload package is used to upload XML file to Web-MIIND [11]. Tomcat is used to deploy the web interface [4]. Other libraries and packages are mentioned in Porting MIIND report, section 6.5 [18]
Eclipse was used as the main developing tool for programming Java.[12]
Linux Shell script was written using vim, as most scripting are done directly on the virtual machine and the cloud test bed [13].
Latex was used for generating this report[6]
SSH protocol was used for handling remote interaction between machines [5].
Cronjob was used for scheduling and synchronising time. [2].

8.4 Appendice C - Ethic issue

No ethic issue was involved in this project.

8.5 Appendice D - Proof of work

8.5.1 XML usage interface

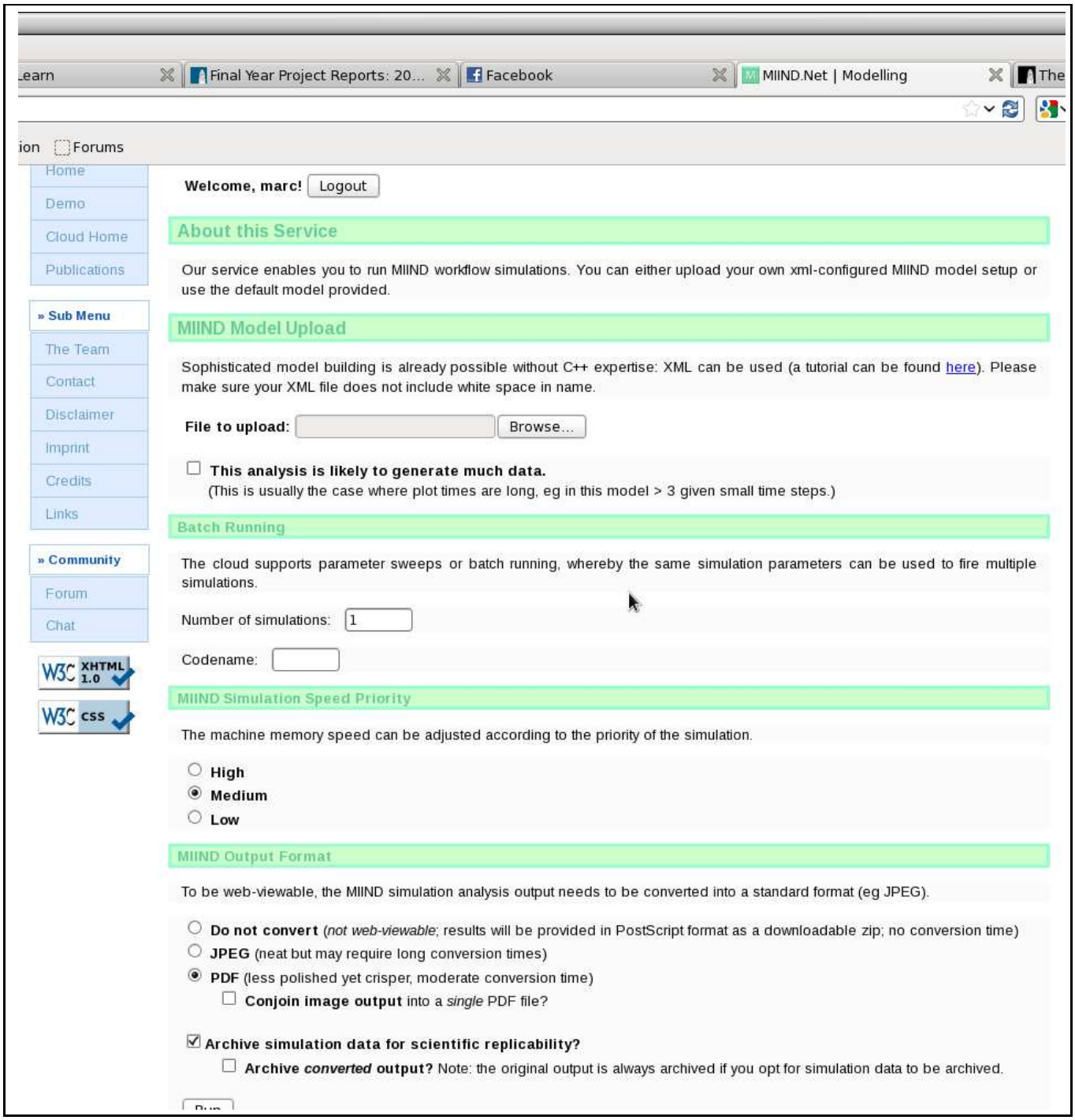


Figure 22: XML usage web interface

8.5.2 WorkFarm

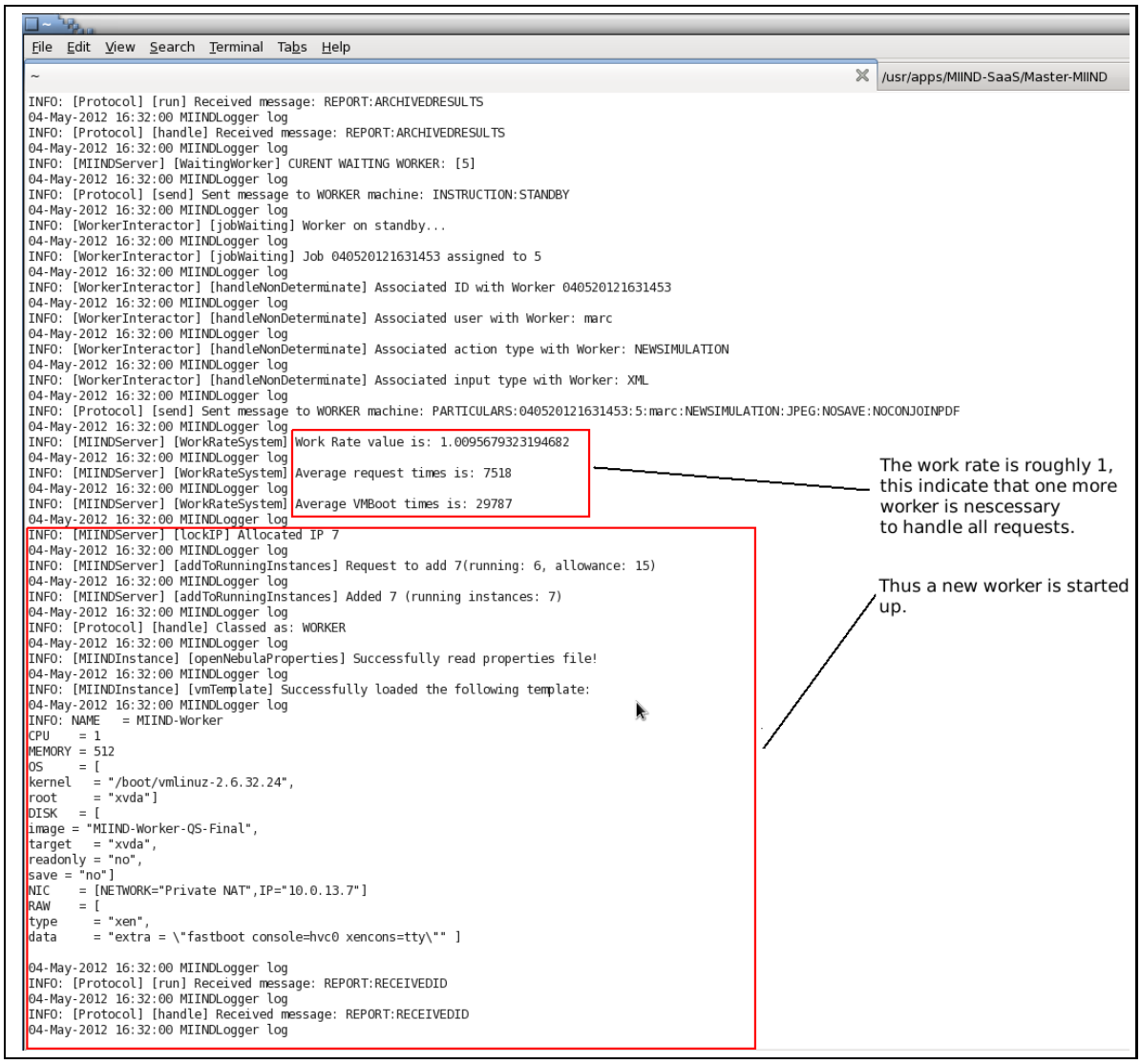


Figure 23: Work Rate System

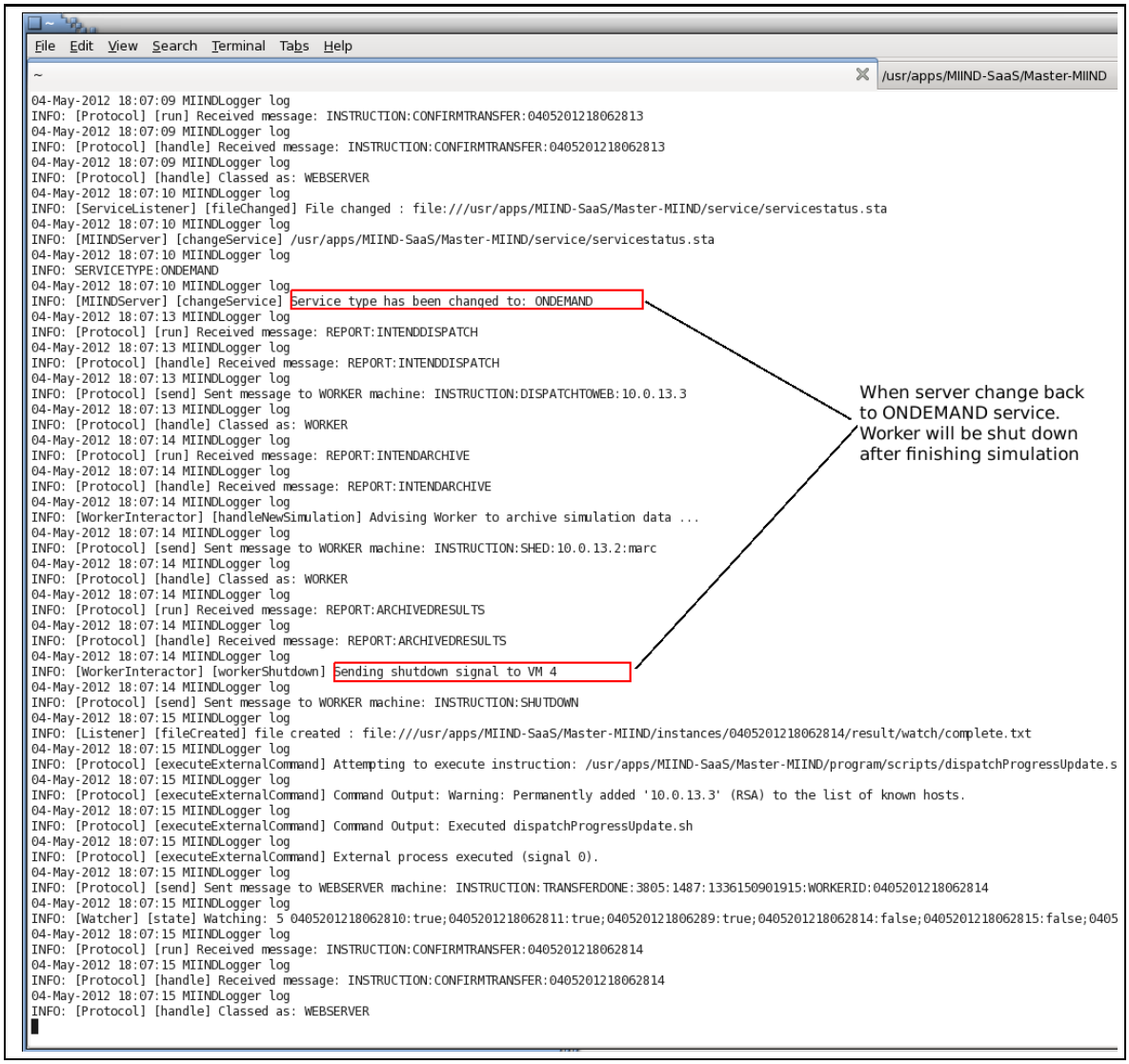


Figure 24: Server changes service type

8.5.3 Evaluation test run

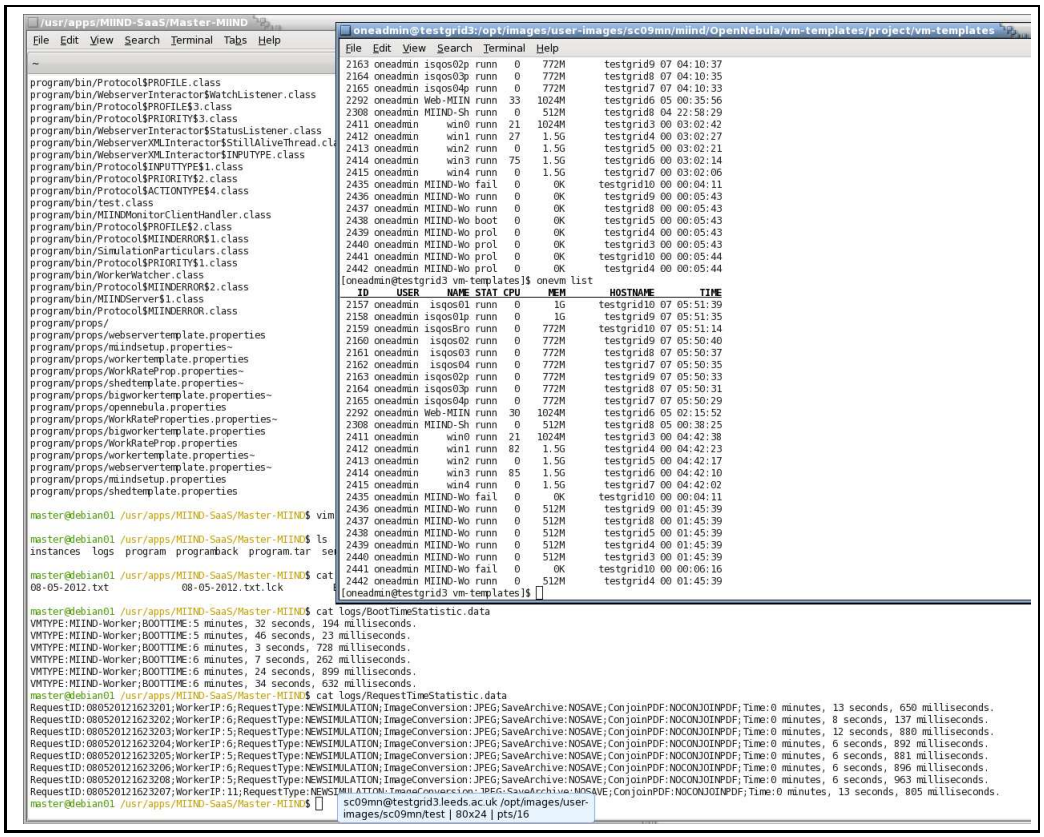


Figure 25: Test run

8.5.4 MIIND Updater

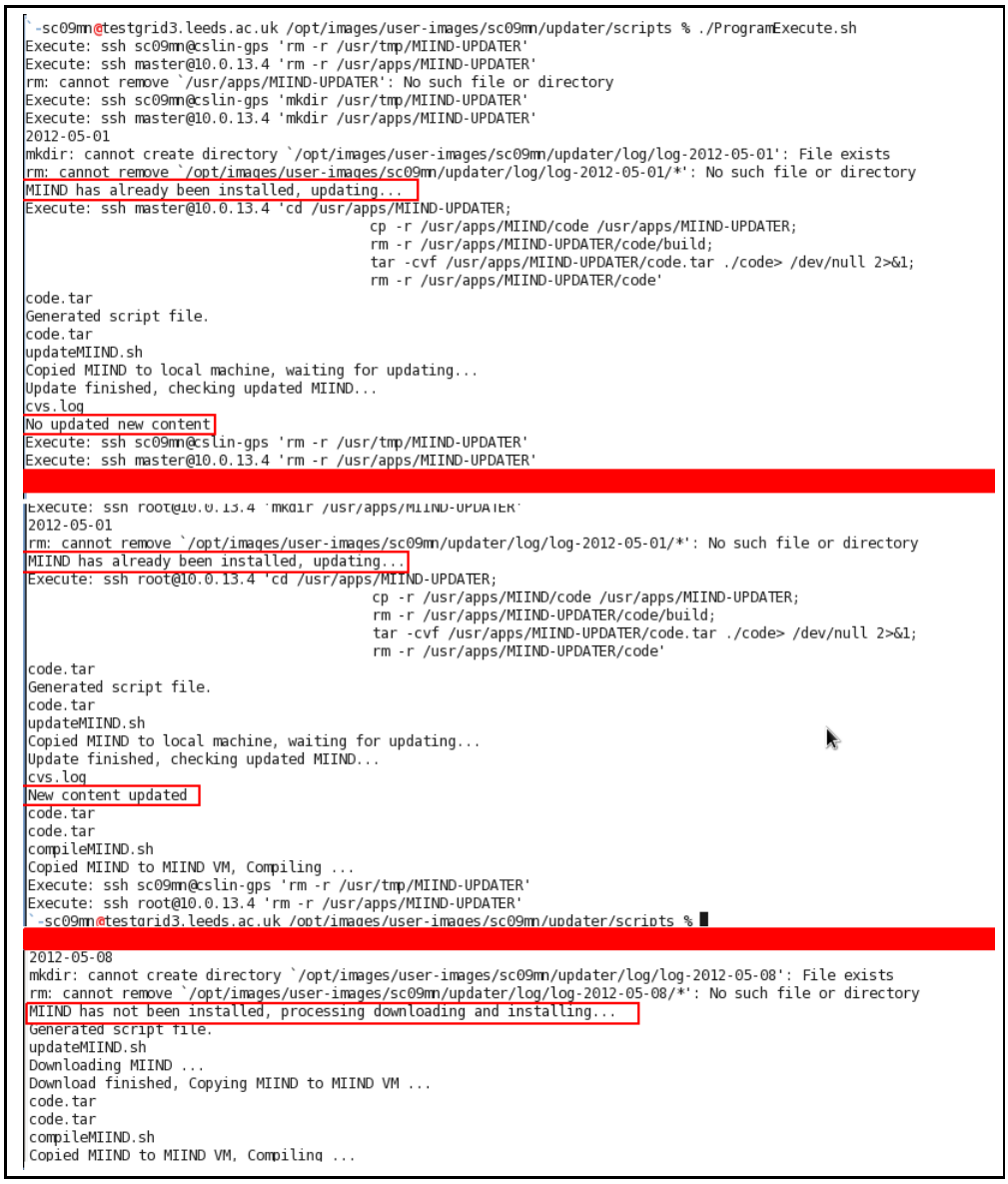


Figure 26: MIIND-updater detects whether new version of MIIND is available

References

[1] Java documents. Available at: **http://docs.oracle.com/javase/1.4.2/docs/api/allclasses-noframe.html.**

[2] Cronjob, 2009, [cited on 4 May, 2012]. Available at: **http://www.thegeekstuff.com/2009/06/15-practical-crontab-examples/.**

[3] MMPI Global Communication Embarrassingly Parallel Problems Simple Load Balancing, 2011 [cited 4 May 2011]. M.E.Hubbard, Lecture slides TC32: Parallel Scientific Computing.

[4] Tomcat, 2011, [cited on 4 May, 2012]. Available at: **http://tomcat.apache.org/tomcat-6.0-doc/index.html.**

[5] SSH Port Forwarding, 21 March, 2011 [cited 4 May 2011]. Available at: **http://www.symantec.com/connect/articles/ssh-port-forwarding.**

[6] Latex guide, 30 April 2012, [cited on 4 May, 2012]. Available at: **http://en.wikibooks.org/wiki/LaTeX**.

[7] A general overview of miind, [cited 4 May 2011]. Available at: **http://miind.sourceforge.net/.**

[8] Opennebula.org, [cited 4 May 2011]. Available at: **http://opennebula.org/documentation:documentation.**

[9] Xen. xen configuration file options., [cited 4 May 2011]. Available at: **http://wiki.xensource.com/xenwiki/XenConfigurationFileOptions.**

[10] Xen hypervisor, [cited 4 May, 2012]. Available at: **http://xen.org/products/xenhyp.html.**

[11] Apache Commons Virtual File System, [cited on 4 May, 2012]. Available at: **http://commons.apache.org/vfs/.**

[12] Eclipse, [cited on 4 May, 2012]. Available at: **http://www.eclipse.org/**.

[13] Vim the editor, [cited on 4 May, 2012]. Available at: **http://www.vim.org/**.

[14] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Jrnl. of Network and Computer Applications*, Volume 23:187–200, 2001.

[15] K. Craig-Wood. Definition of cloud computing, incorporating nist and g-cloud views. 24 February, 2010. Available at: **http://www.katescomment.com/definition-of-cloud-computing-nist-g-cloud/**.

[16] M. de Kamps et al. The state of miind. *Jrnl. Neural Networks*, pages 1164–1181, 2008. Available at: **www.elsevier.com/locate/neunet**.

[17] I. Foster et al. Cloud computing and grid computing 360-degree compared. *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, 2008.

[18] B. Gerckens. Cloud computing for computational biology: Porting miind. 2011. Available at: **http://www.comp.leeds.ac.uk/fyproj/previous-titles/bsc2010.html**.

[19] M. Jung, S. Loureiro, and F. Donnat. Is cloud-computing centralized or decentralized?, Part 4. February 9, 2011. Available at: **http://elastic-security.com/2011/02/09/is-cloud-computing-centralized-or-decentralized-part-4/**.

[20] J. D. Meier. Software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). 11 February, 2010. Available at: **http://blogs.msdn.com/b/jmeier/archive/2010/02/11/software-as-a-service-saas-platform-as-a-service-paas-and-infrastructure-as-a-service-iaas.aspx**.

[21] J. D. Myerson. Cloud computing versus grid computing. 03 March, 2009. Available at: **http://www.ibm.com/developerworks/web/library/wa-cloudgrid/**.

[22] A. Rosenthal et al. Cloud computing: A new business paradigm for biomedical information sharing. *Journal of Biomedical Informatics*, Volume 43:342–353, April 2010. Available at: **http://www.sciencedirect.com/science/article/pii/S1532046409001154**.

[23] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Computer Communication Review*, Volume 39:50–55, January 2009.