FightMalaria@Home

Distributed biological screening through Volunteer Computing to find useful protein/drug matches for the purpose of malarial drug research

Peter Lavin, Neil Whelan

Thesis presented to the University of Dublin,
Trinity College in partial fulfilment of the requirements for the
Degree of Master of Science in Computer Science

Department of Computer Science,
University of Dublin, Trinity College

September 2007

Declaration

I declare that the work in this dissertation is, unless otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university

Signed: _____

Peter Lavin

14th September 2007

Declaration

I declare that the work in this dissertation is, unless otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university

Signed: _____

Neil Whelan

14th September 2007

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____

Peter Lavin

14th September 2007

Permission to lend and/or copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____

Neil Whelan

14th September 2007

Acknowledgements

Firstly, we would like to thank our supervisor, Dr. Brian Coghlan, for his support and attention throughout this project. We would like to thank Dr. Eamonn Kenny for his advice and technical guidance, and Dr. David O'Callaghan for providing valuable technical assistance. We would also like to thank Dr. Anthony Chubb from the Royal College of Surgeons in Ireland for his vision in proposing this project.

Finally, we would like to thank our friends and families, without whom this project would not have been possible.

Peter Lavin, Neil Whelan.

Abstract

Malaria infects up to 650 million people in the world each year, resulting in between 1 to 3 million fatalities. Up to this point, attempts to find a cure or a treatment to combat infection have been obstructed as malaria has continually mutated to counteract the effectiveness of each drug that has been applied. As conventional wet-lab methods of drug development are currently not yielding a conclusive cure, the area of bioinformatics is a viable area of research for simulating drug development. The widespread availability of computing resources offers the opportunity of applying a more brute-force approach to the issue of sourcing a malarial cure.

Ever more sophisticated software is being developed for the accurate screening of biological data in the form of digital representations of the chemical constitution of drugs and proteins. An example of such software is eHiTS (electronic High Throughput Screening). eHiTS software uses state-of-the-art 3D modelling to accurately re-enact the chemical bindings that occur between drugs and protein.

The concept of Volunteer Computing represents an attempt to harness the potentially vast unused computing resources of the Internet. In this manner, volunteers on the Internet can donate processing time on their PCs to the administrators of scientific projects who may not have the resources to equal the processing power that this unused computing resource can provide. Berkeley Open Infrastructure for Network Computing (BOINC) is an open-source middleware platform designed for the implementation of Volunteer Computing projects.

This dissertation describes the planning, implementation, and evaluation of an eHiTS-enabled BOINC Volunteer Computing project that is accompanied by an extensive database repository of digitized drug and malarial genome datasets, a processing management system for dispatching jobs to the BOINC project, and an analysis system for assessing the results of large-scale biological screening.

The architecture of the project is designed with a view to being integrated into a Grid Computing environment, where processing jobs would ultimately be sent indiscriminately to either Grid resources, or to the BOINC Volunteer Computing project.

Division of labour between the two students

While there was some background reading and research which was common to the two students, the design and implementation was divided into two sections.

Peter Lavin

The work in this section consists of an examination of the existing formats of chemical and biological data which was relevant to this work. It also looks at existing grid and storage resources and the parameters and constraints around submitting individual units of work to such a resource. An evaluation of contemporary drug screening software programs and their inputs and outputs was also carried out. This work was eventually extended to modifications to one screening program to customise it specifically for use in grid and other restricted environments. Regard was given to the influence of all these entities on the design of flow of data to worker nodes and processing of results returned for such a large drug screen challenge. In this work, this process is known as the Master Process. It is designed to be adaptable for resources on a grid element or on a remote BOINC client. Considerations were effectiveness, manageability and systematic distribution. Paramount importance was also given to the usefulness and integrity of the results recovered form worker nodes.

Neil Whelan

The main component of this section of the overall project was the configuration and implementation of a Volunteer Computing project that was developed with the BOINC middleware platform. This BOINC-based project enables biological screening software to be executed on the BOINC clients. This facility was provided by integrating code into the BOINC environment by way of BOINC API functions. Additional functionality enables the modules that create processing jobs to be invoked remotely, and the files that result from computation on the BOINC clients to be transferred to a remote location.

Table of contents

Article I. Introduction……...12
Section 1.01 Grid Computing 13
Section 1.02 Grid Implementation and Management 15
Section 1.03 Submitting work or jobs to a grid 15
Section 1.04 Job Description Language..... 16
Section 1.05 Volunteer Computing..... 17
Section 1.06 Desktop Computing 18
Section 1.07 Volunteer Computing vs. Grid Computing..... 18
Article II. State of the Art……..21
Section 2.01 WISDOM 21
Section 2.02 Commercial SUN and Google Grid Utilisation 22
Section 2.03 BOINC 22
Section 2.04 Related BOINC projects 25
Section 2.05 BOINC on Grids 26
Article III. Technologies used (I)……28
Article IV. Technologies used (II)……...29
Article V. Design considerations……...30
Section 5.01 Description of File Types..... 30
Section 5.02 Protein Structures..... 30
Section 5.03 Ligand Structures 31
Section 5.04 Docking Energy Between Structures 32
Section 5.05 File Formats for Structures 32
Section 5.06 SMILES Strings 33
Section 5.07 Screening Software 33
Section 5.08 Visit to Royal College of Surgeons Ireland. 33
Section 5.09 eHiTS Installation. 34
(a) License Granting 35
(b) eHiTS Program Operation 35
(c) Fast and Light Strategy to Screening 36
(d) Comparison of AutoDock and eHiTS 37
Section 5.10 Estimation of Time Required for challenge 38
(a) Relationship of Receptor File size to Docking Time 38
(b) Relationship of Ligand File size to Docking Time 39
Section 5.11 Preparation of Ligand Files from ZINC Database 40
(a) Naming scheme for ligands Files..... 40
(b) Existing Naming Scheme in ZINC Database..... 41
(c) SDF Tags..... 41
Section 5.12 Quantification of Data Storage Requirements 42
Section 5.13 Database Design Consideration 43
Article VI. Overview of BOINC……45
Section 6.01 Architecture of a BOINC server 45
(a) BOINC tools and libraries..... 46
(b) MySQL database..... 47
(c) Apache Web Server..... 47
Section 6.02 BOINC project 48
(a) Directory structure 48

(b) Database 50
(c) Configuration file 51
Section 6.03 Further tools and configuration files 52
(a) Project.xml 52
(b) Config.xml 53
(c) Upload/Download directory structure 55
(d) Schedulers.txt 56
Section 6.04 Security 57
Section 6.05 Applications 57
(a) Application API functions 58
(b) Adding applications and platforms 59
(c) Updating the version 59
Section 6.06 Generating work 60
(a) Preparing input files 60
(b) Creating a work-unit 61
Section 6.07 XML Templates 62
(a) Work-unit template 63
(b) Result template 64
Section 6.08 BOINC web-page 65
Section 6.09 Screensaver graphics 66
Section 6.10 BOINC daemons 66
(a) Feeder 67
(b) Transitioner 67
(c) File Deleter 67
(d) Database Purger 67
(e) Work Generator 68
(f) Validator 68
(g) Assimilator 68
Section 6.11 BOINC Client 68
(a) Directory structure 69
(b) Work-unit template 69
(c) Results template 70
(d) Doing work 70
Section 6.12 Life cycle of a work-unit 71
(a) Creating a work-unit 71
(b) Downloading a work-unit 71
(c) Processing a work-unit 72
(d) Uploading results 72
(e) Back-end processing 72
Article VII. Implementation (I) 73
Section 7.01 Python 2.4 setup for Master Process 73
(a) Python 2.4 73
(b) OEChem Software Licensing and Installation 74
(c) Other packages Installed 75
Section 7.02 Inserting Naming Scheme Data 75
Section 7.03 Insertion of SMILES Strings 76
Section 7.04 Preparation of Receptor files 77
Section 7.05 Changes to eHiTS program for security of distribution 78
Section 7.06 Database Changes made to Accommodate the eHiTS changes 79

Section 7.07 Using Tags with tmb binary files 80
Section 7.08 Database schema 80
(a) Details of tbl_prot_list table 81
(b) Details of tbl_ligfiles table 82
(c) Details of tbl_b100_files table 82
(d) Details of tbl_<*prot_name*>_jobs table 84
(e) Details of tbl_<*prot_name*>_results table 85
Section 7.09 Rationale for Master Process 86
(a) Design Considerations for the Master Process 86
(b) ProtoPlanner Daemon 87
(c) ProtoResults Daemon 89
(d) Updating of comp_1 and comp_2 in table tbl_<prot_name>_jobs 90
(e) Interaction with the Database 90
Section 7.10 Configuration file and access to it 91
Article VIII. BOINC Implementation 93
Section 8.01 Application platform 94
Section 8.02 Application version 95
Section 8.03 Application code 96
(a) Command line arguments 96
(b) BOINC initialization and logging 97
(c) Directory change 97
(d) Extracting eHiTS 98
(e) Licensing and eHiTS modification 98
(f) Extracting the ligand and receptor files 99
(g) Running eHiTS 99
(h) The eHiTS results file 100
Section 8.04 Work-unit template 100
Section 8.05 Result templates 101
Section 8.06 Work generator 103
(a) Command line arguments 103
(b) Environment switch 103
(c) Work-unit definition 104
(d) API call to create work 104
Section 8.07 Move files 105
Section 8.08 Daemons 106
(a) Validator 106
(b) Assimilator 107
Section 8.09 SCP configuration 107
Article IX. Problems Encountered (I) 109
(a) File Handling with Large Number of Files 109
(b) Maximum Packet Size for Transactions with Database 109
(c) Loading binaries to the database 110
(d) Slash Characters in Protein Files 110
(e) Linux 111
(f) Compatibility of Python versions and libraries 111
Article X. Problems Encountered (II) 112
Section 10.01 Linux 112
Section 10.02 Issues with BOINC 112
(a) Documentation 113

(b) Documentation - Calling proprietary software in BOINC client 113
(c) Error-handling 114
Section 10.03 eHiTS performance on BOINC 115
Section 10.04 Conclusion 115
Article XI. Analysis (I) 116
Section 11.01 Comparison to Initial Goals 116
Section 11.02 Suitability of work units 116
Section 11.03 Speed of dispatching of work units 117
Section 11.04 Comparison to past drug screens 117
Section 11.05 Direct use of a grid environment 117
Section 11.06 Master Process Database Performance 117
Section 11.07 eHiTS 118
Article XII. Analysis (II) 119
Section 12.01 Success of initial goals 119
Section 12.02 Consistency of results 120
Section 12.03 Choice of software 120
Section 12.04 Applicability of project architecture 121
Section 12.05 Performance of BOINC 121
Section 12.06 BOINC security 122
Section 12.07 Changes to BOINC model 122
Section 12.08 How project compares to related work 123
Section 12.09 Suitability of interface 124
Section 12.10 Conclusion 124
Article XIII. Future Work (I) 126
Section 13.01 Modifications to eHiTS Program 126
Section 13.02 Social Grid Agents 126
Section 13.03 Security around Planner Process 127
Section 13.04 Flexibility around the size of block files per work unit 128
(a) Changes Driven By Receptor Size 128
(b) Changes Driven By Available Resources 128
(c) Areas of change to Optimise Block file size 128
Section 13.05 Changes to structure of eHiTS for Reusing Pre-process data 129
Section 13.06 Speed of Master Process 130
Section 13.07 User interface for Interested Parties 131
Article XIV. Future Work (II) 132
Section 14.01 FightMalaria@home web-page 132
Section 14.02 Project security 132
Section 14.03 Project screensaver 133
Section 14.04 Locality scheduling 133
Section 14.05 Porting BOINC project to Grid 134
Section 14.06 Change from system call paradigm 134
Section 14.07 Setup of Globus Toolkit container 134
Section 14.08 Research related projects 135
Section 14.09 Re-evaluate screening software 135
Section 14.10 Configure project code for different platforms 136
Section 14.11 Increase functionality of Validator 136
Section 14.12 Consider retaining some aspects of existing implementation 137
Section 14.13 Implement PayPal utility 137
Section 14.14 Publish research paper 137

Article XV. Conclusions (I)……139
Article XVI. Conclusions (II)……...140
Article XVII. Appendix 1……..141
Article XVIII. Bibliography……143

Table of Figures

Figure 1 - Topology of a typical Grid 13
Figure 2 - Geographical diversity of Grid sites 14
Figure 3 - BOINC logo 22
Figure 4 - Choice of BOINC projects 23
Figure 5 - SETI@home screenshot 24
Figure 6 - Account Manager screenshot 25
Figure 7 - Screenshot of PrimeGrid web-page 26
Figure 8 - Graphical representation of a Protein 31
Figure 9 - Graphical representation of a Ligand 31
Figure 10 - BOINC architecture 46
Figure 11 - Database tables 81
Figure 12 - Technical architecture of project 86
Figure 13 - FightMalaria@home architecture 94
Figure 14 - Master Process interaction with a Grid environment 127

Table of Tables

Table 1 - File formats for molecular structures 32
Table 2 - Comparison of times for docking of different Receptors against ligands 39
Table 3 - Storage requirements 43
Table 4 - Receptor database table 81
Table 5 - Ligand database table 82
Table 6 - Database table of ligand blocks 83
Table 7 - Database table of work-units for each protein 84
Table 8 - Database table for protein results 85
Table 9 - Sample content from table 8 85

Article I. Introduction

Malaria is considered a somewhat forgotten disease. Although, not as high profile in the media as HIV and AIDS, it is secondary to HIV and AIDS only due to scale of that problem on the African continent. Malaria is equally debilitating in human and economic terms. Despite this, it does not get attention and investment in research in drug development it might deserve from pharmaceutical companies.

The genomic sequence of the mosquito borne protozoan which spreads the disease (namely Plasmodium Falciparum) has been identified for some years now. Automated 3D modelling of the molecular structures of these proteins has lead to the availability of the 400 or so proteins that can be considered the malaria disease.

There are approx $25x10^6$ identified drug compounds (ligands). 3D molecular structures in the form of ASCII text files for these compounds are freely available from drug databases such as ZINC, MOE and DrugBank (mainly through academia). The overall aim and final output of this project is to develop a distributed and parallel drug screen. For each of the 400 proteins, this screening work aims to yield a relatively short list of ligand or drug compounds which can be considered *leads* in the search for a suitable drug for the disease. A ligand or compound can be considered a *lead* if a high docking energy is recorded when it is screened or docked against that protein. This high docking energy indicates that the drug compound interacts with the protein in such a way that it will inhibit its behaviour and interactions in a biological system. In the human body, this may possibly stop the harmful action of the protein. The aim is to generate a list of roughly 30-50 promising drug compounds for each protein. When these relatively short lists of high scoring ligands are assembled for the various malarial proteins, they can be presented to various pharmaceutical companies for further testing, validation and possibly development and eventually production at their laboratory facilities.

Section 1.01 Grid Computing

Grid computing is a model for allowing institutions such as universities, large corporation and organisations to share and use large numbers of computers as a single resource[1]. This resource is suitable for application to a scientific or technical problem such as ongoing climate change calculations or drug research. Typically, such challenges will require large numbers of processing cycles and storage capacity. Individual CPUs carry out computations on such a problem in parallel with other CPUs in the grid. Fig 1 outlines the topology of a typical grid.

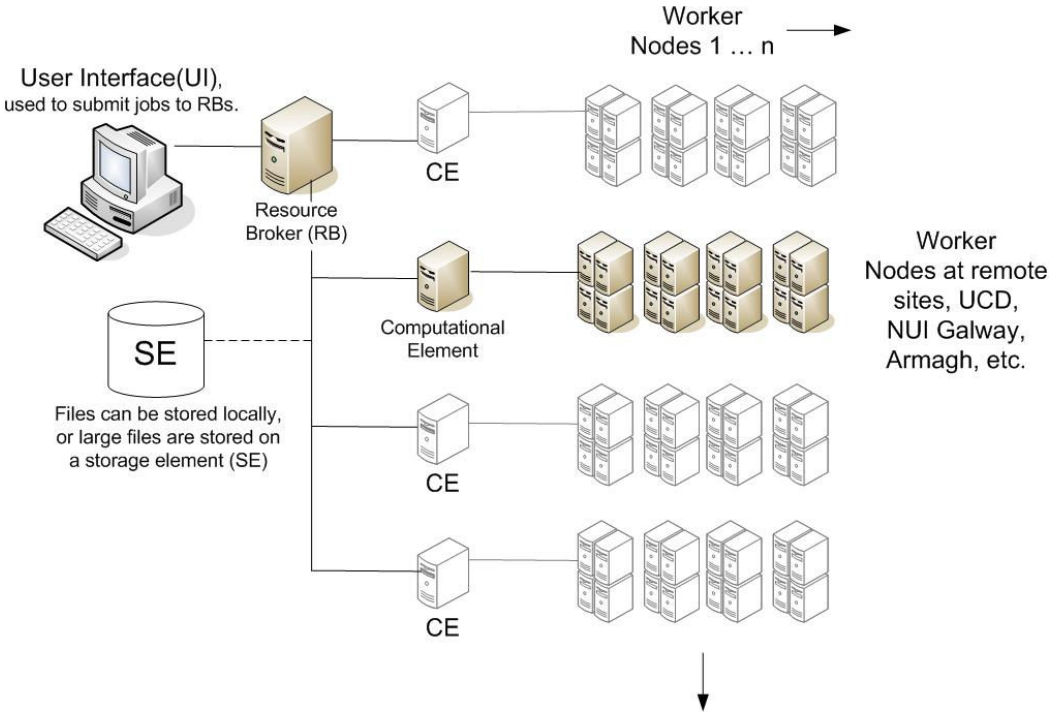


Figure 1 - Topology of a typical Grid

In providing clustering of remotely distributed CPUs, a site consists of a cluster of CPUs at a particular location. This will be linked to similar clusters which may be at geographically dispersed locations. Each cluster has one Computational Element (CE) as the head node of a site. It is this node which is accessed a Resource Brokers (RB) when work is being disseminated. An example of this topology is Grid-Ireland which has locations in six sites in the Dublin area and a further 10 in Ireland and Northern Ireland. Fig 2. is a screen shot of a real-time monitor for conditions of various grid sites in Ireland.

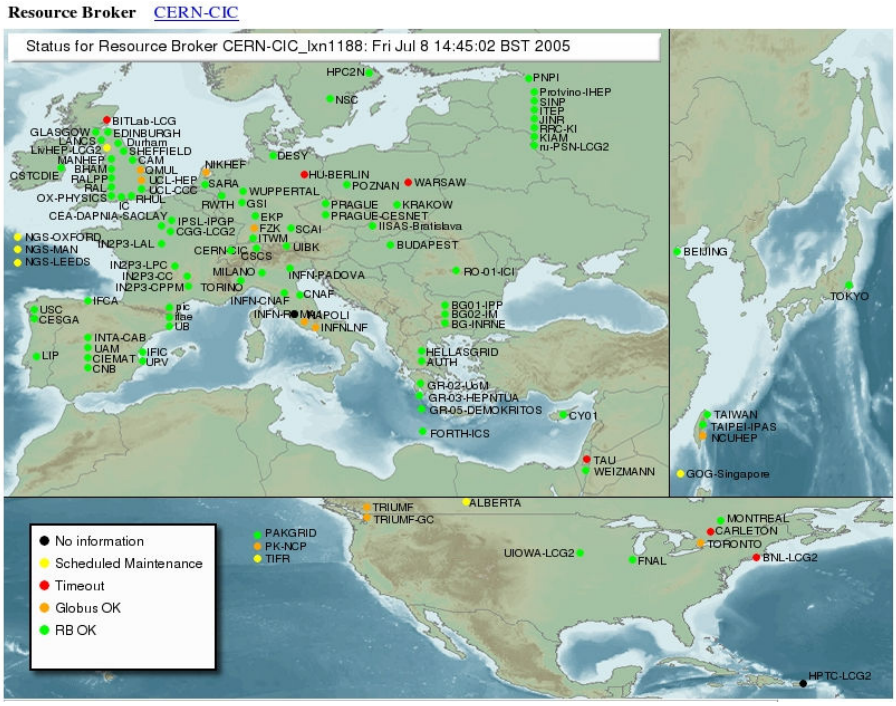


Figure 2 - Geographical diversity of Grid sites

Section 1.02 Grid Implementation and Management

Purpose build software packages such as Globus and Condor are used to manage grid resources. Globus software was developed in the 1990s[2]. It uses existing standards such as SSL/TLS, X509 Proxy Certificates, SOAP, HTTP and some more to implement and create grid systems. Occasionally, grid resources can comprise of under utilised hardware in labs and offices. Such systems may use CPU scavenging software to access computational cycles.

The grid infrastructure at Trinity College Dublin, which is part of Grid-Ireland consists of dedicated racks of computer hardware. Virtual machine (VM) software3 is used for optimising and fully exploiting this hardware. VM technology allows multiple instances of an operating system to run on one computer and reduces costs associated with hardware, personnel and space at grid sites.

Section 1.03 Submitting work or jobs to a grid

A work unit or grid job for submission to a grid consists of an executable file, input file(s) and the required output file. Prior to submission, if the input files are small, they can be stored on the user interface machine (UI). Larger files may be stored on a storage element (SE) which will make them accessible to the worker nodes (WNs). Jobs are submitted from the UI to a resource broker (RB). Access to the RB from the UI is authenticated using an X509 proxy certificate. Access is further controlled by the requirement of a user to be a member of a Virtual Organisation (VO). Short term X509 proxy certificates are issued for 12 hours (typically). In the event of a proxy certificate being compromised, any negative impact will be kept to a minimum by this measure. A RB has access to computational elements (CEs) at the head of a cluster at a grid site. As outlined, CEs and clusters will be located at different sites.

Section 1.04 Job Description Language

Globus provides Job Description Language (JDL) to define the parameters required to submit work to a RB. An example of a JDL file is outlined here. The executable, its arguments and the name and location of the output file are defined by this.

```
[
  Type = "job";
  JobType = "Normal";
  Executable = "eHiTS.sh";
  Arguments = "-ligand b_1_a.tmb – receptor 1a07.tmb";
  VirtualOrganisation = "";
  StdOutput = "sim.out";
  StdError = "sim.err";
  InputSandbox = {"eHiTS.sh"};
  OutputSandbox = {"sim.err", "sim.out"};
  OutputData = {
    [
      Outputfile = "best_scotes.txt";
      LogicalName = "lfn:/grid/gitest/best_scotes.txt";
      StorageElement = "gridstore.cs.tcd.ie";
    ]};
  Requirements=(other.GlueCEInfoTotalCPUs>0);
  Rank=(other.GlueCEStateFreeCPUs);
  RetryCount = 0;
]
```

Section 1.05 Volunteer Computing

The idea of Volunteer Computing was conceived in 1996 with the advent of the Great Internet Mersenne Prime Search (GIMPS) project. Its goal is to obtain the voluntary donation of computing resources from Internet users. Achieving this aim provides projects with access to large numbers of processing cycles without having to commit extensive financial resources.

Many other projects followed the lead of GIMPS to result in a steady increase in the popularity of Volunteer Computing. The media and processing peak was reached in 1999 when Berkeley University's SETI@home managed to attract several hundred thousand volunteers. Some business ventures were also started with the objective of using the Volunteer Computing model; however Volunteer Computing has predominately been associated with non-profit scientific research.

Initially, the project software itself consisted of monolithic modules, where the computation and distributed computing code were combined into a single application. This model soon gave way to middleware platforms where the volunteers downloaded light-weight clients that were independent of the scientific computation that was being executed. The most widely-used of these middleware platforms is the Berkeley Open Infrastructure for Network Computing (BOINC). It was developed by Berkeley University as an abstraction of SETI@home.

The template model for most Volunteer Computing platforms is a client/server model where clients poll project servers over the Internet requesting jobs for download and processing. The clients upload results to the project servers whenever the processing has completed. This client-“pull” model is applicable to an Internet environment as many volunteer nodes are behind firewalls, and hence unable to accept incoming connections from project servers.

As volunteer are unaccountable for the quality and reliability of the processing results that

they upload, problems inevitability arise with regard to ensuring that a complete set of valid scientific results is achieved. Among these issues are the heterogeneity of volunteer machines on which the clients are to be run, their intermittent availability, and their occasional malfunction. There is also the risk of disruption due to malicious users falsifying results. The most common method used to counteract such issues is termed redundancy whereby each processing job is sent to multiple for processing. The repetition of a common result ensures, with reasonable certainty, that a correct result is received.

[Section 1.05 sourced mainly from [4]]

Section 1.06 Desktop Computing

The concept of Desktop Computing follows closely on that of Volunteer Computing. The same client/server, client-"pull" model is usually employed to adhere to and take advantage of platforms that are primarily designed for use in Volunteer Computing projects. However, the hosts – typically Desktop PCs - used for the processing are located within the organisation that is running the project. Inter-organisational desktop pooling is also within the scope of Desktop Computing. Parallels can be drawn between Desktop or Volunteer Computing and Grid Computing, where especially dedicated processing resources are utilised either within an organisation or pooled in an inter-organisational arrangement. There are also some key differences which are discussed in the next section

[Section 1.06 sourced mainly from [5]]

Section 1.07 Volunteer Computing vs. Grid Computing

The differences between Volunteer and Grid Computing are often confused, and sometimes the two principles are thought to be more or less the same. However, there are several properties that clearly separate the two concepts into different genres.

Firstly, both the operator and the volunteers of a Volunteer Computing project have to use the infrastructure, and pay the bills, of an Internet Service Provider (ISP). Therefore, careful planning is required for data-intensive applications to maximise the use of the network bandwidth. However, the developers of a Grid Computing system are free to use their own resources as they see fit. Another difference arises in that the actual volunteers may possess little IT knowledge, and so Volunteer Computing software must be simple to install and cannot depend on specific operating systems, or unconventional platform configurations. It must also be self-healing, and must not be over-intrusive to the owners of volunteered resources. This is not the case in Grid Computing, where skilled IT developers are intimately familiar with the computing system. A further distinction applies where a volunteer user downloads client software that interacts with the server configuration, and permits it to connect to the Internet. While personal details may be requested from the volunteer for profiling purposes, these cannot be made to be a requirement. The details provided cannot be verified or authenticated. A Grid infrastructure is connected over a private medium, and the identity of the owner and user of a resource is known.

As described in section 1.02, client-"pull" architecture is needed to sidestep the issue of Internet firewalls blocking attempts by the server to dictate the actions of clients. Any architecture is possible with Grid Computing, the design of which depends on the goals of the system. Volunteer Computing is asymmetric, where volunteers supply computing resources to projects without anything in return, other than personal satisfaction. Conversely, Grid Computing is symmetric, where one organisation might borrow the resources of a second organisation for a period of time, on the condition that the favour can be returned in the future.

Volunteers usually operate anonymously, and cannot be held responsible for the quality of processing results, or for problems caused by misuse of the server. Volunteered computer resources are not under the control of the administrators of a Volunteer Computing project, and it is a matter of trust that the majority of users have attached to a project with honourable intentions. Volunteer Computing projects may be subjected to the same attacks as any Internet application, such as denial of service. Malevolent users may also return falsified results to corrupt or skew the overall validity of computational results. They may

even attempt to claim processing credits for work that they have not done, thus gaining more trust from the project. Typically, systems are put in place to counteract such attacks. Redundant computing will yield the correct result through consensus, and also provide a basis for the detection of credit falsification.

On the other hand, Grid Computing environments are usually protected from external attack by employing stringent security measures. There is generally a very strict access process for users of a Grid, and although organisations may often share resources, they often have common security interests. Therefore, processing can proceed in such environments unhindered by the treat of attack. Much higher rates of processing efficiency are achieved with Grid Computing as redundancy does not need to be employed. The integrity of processing resources can be guaranteed.

Finally, public relations are needed to attract enough volunteers for a Volunteer Computing project to become successful. The achievement of sufficient processing power in Grid Computing usually hinges on the extent of the financial backing that is available.

However, the continuing validity of some of the contrasts that have been outlined in this section may be called into question by recent developments in which some organisations, such as Sun Microsystems Inc., have offered their Grid resources to the market at a flat-rate tender. This topic is discussed in more detail in section 2.02.

[Section 1.07 sourced mainly from [6]]

Article II. State of the Art

There have been many successful grid deployment of drug screening software. Some of there are outline here. To some degree, this project replicated some work already done, but aims to be free of the constraints of limited grid resources and competition for these resources. The real world reality is that any initiative on drug development for Third World countries has to compete with material modellers, astronomers, etc. Details are outlined here of some similar examples were drug screening was carried out on grid resources and the quantities of work done.

Section 2.01 WISDOM

In 2005, the WISDOM initiative (Wide In Silico Docking on Malaria) ran on EEGE[7] and docked over 42 million drug compounds against a single receptor. This challenge ran for approximately 37 days. It utilised over 1000 computers in over 15 different countries around the world[8]. Although the size of this screen was significant, it docked compounds against only one of the approximately 400 receptors known in the malaria disease. However, it did demonstrate that screens of this size were feasible, and also lowered barriers to future engagements of this size. In 2006, a further challenge was undertaken, this time docking 1 million compounds against 5 receptors[9]. For this work, FlexX[10] and AutDock were deployed. Using the EGEE grid infrastructure for a malaria screening campaign, WISDOM have demonstrated that resource sharing on an eScience infrastructure such as EGEE provides a new model for doing collaborative research to fight disease.

The name WISDOM has now become a generic name for grid based drug screening on EGEE. In April 2007, a collaboration of Asian and European laboratories docked 3,000,000 compounds against 8 different structures of the avian flu virus H5N1[11] using WISDOM. In terms of storage, over 60,000 files were produced and 600 GB of storage space was required.

The challenge proposed here contains roughly 400 receptors and proposes to eventually screen these against a similar or larger number of drug compounds as have been used in the above work. A significant difference is that this work intends to utilise the combined CPU cycles of freely available volunteer computer capacity and grid resources.

Section 2.02 Commercial SUN and Google Grid Utilisation

Among others, both Sun and Google provide grid resources[12]. This is a commercially based venture and priced in 2007 are around $1 per CPU per hour. Although available to any interested party, this resource is most suited to commercially drive small to medium size challenges and where clients do not have access to large IT budgets or expertise.

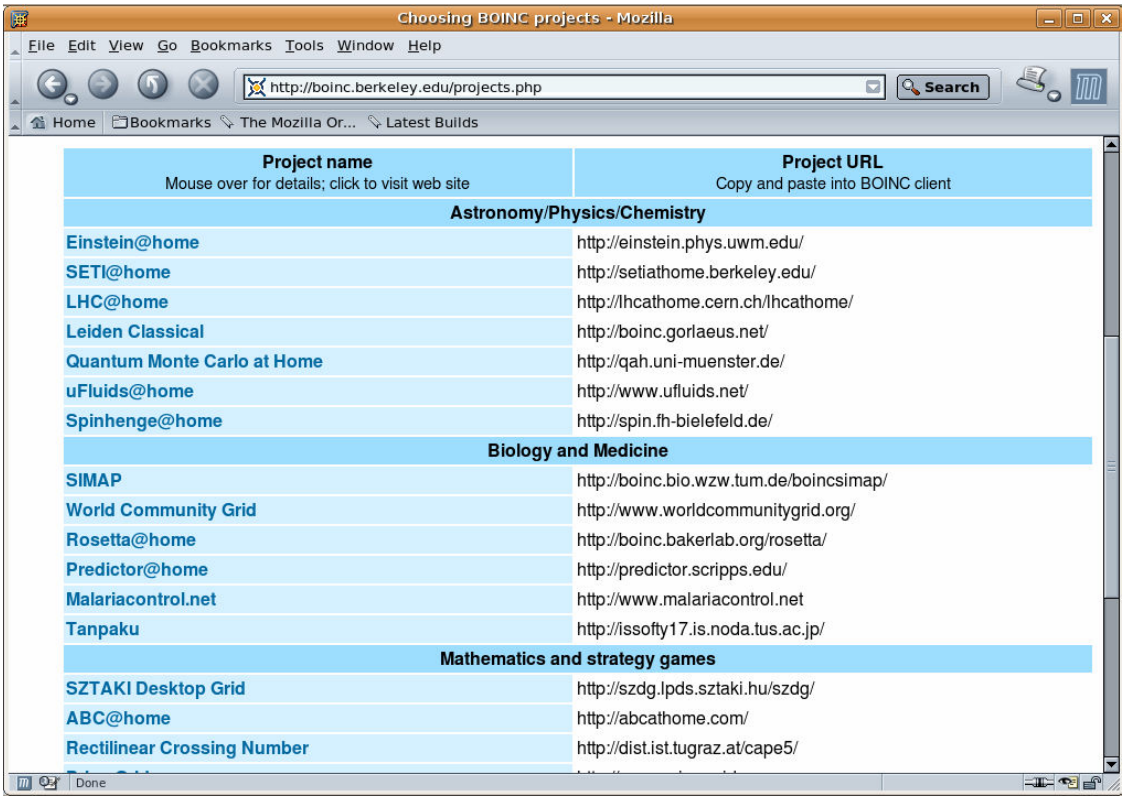
Section 2.03 BOINC

The following sections detail the work that has been done in the area of Volunteer Computing, and more specifically in the area of the BOINC platform, that more directly relates to this project. BOINC projects that may also contribute to malarial drug research are also mentioned. Furthermore, the research that has been done to integrate BOINC with Grids is explained.

Figure 3 - BOINC logo

The BOINC middleware platform is proving to be the method of choice for most scientific projects in need of large amounts of processing resources, especially when only limited resources are available to these projects. As of August 6 2007, BOINC had over 435,000

active computing hosts worldwide that could provide an average of 557 teraFLOPS of processing power[13]. This places the collective processing powers of BOINC ahead of those of the worlds leading supercomputers - IBMs BlueGeneL has 360 teraFLOPS[14]. However, due to the inherent network delays and processing redundancy implicit in making the BOINC model viable, BOINC is commonly referred to as only being a "quasi-supercomputing" platform[13], and does not achieve the levels of efficiency of a supercomputer.



Project name Mouse over for details; click to visit web site	Project URL Copy and paste into BOINC client
Astronomy/Physics/Chemistry	
Einstein@home	http://einstein.phys.uwm.edu/
SETI@home	http://setiathome.berkeley.edu/
LHC@home	http://lhcathome.cern.ch/lhcathome/
Leiden Classical	http://boinc.gorlaeus.net/
Quantum Monte Carlo at Home	http://qah.uni-muenster.de/
uFluids@home	http://www.ufluids.net/
Spinhenge@home	http://spin.fh-bielefeld.de/
Biology and Medicine	
SIMAP	http://boinc.bio.wzw.tum.de/boincsimap/
World Community Grid	http://www.worldcommunitygrid.org/
Rosetta@home	http://boinc.bakerlab.org/rosetta/
Predictor@home	http://predictor.scripps.edu/
Malariacontrol.net	http://www.malariacontrol.net
Tanpaku	http://issofty17.is.noda.tus.ac.jp/
Mathematics and strategy games	
SZTAKI Desktop Grid	http://szdg.lpds.sztaki.hu/szdg/
ABC@home	http://abcathome.com/
Rectilinear Crossing Number	http://dist.ist.tugraz.at/cape5/

Figure 4 - Choice of BOINC projects

BOINC software is open-source, and is released under the GNU Lesser General Public License. This fact, combined with the processing advantages that BOINC offers, has resulted in 14 projects being in operation, and also approximately 33 more being in development as of July 5, 2007[13]. Both command line and graphical user interfaces (the Account Manager) are used to administer the BOINC client. The Account Manager and the BOINC core client communicate over RPC. This means that the Account Manager can potentially be used to remotely control a farm of operating core clients. A graphical

screensaver can also be implemented by BOINC projects. Typically, the graphics display the progress of processing on the BOINC client machine as it occurs in real-time.

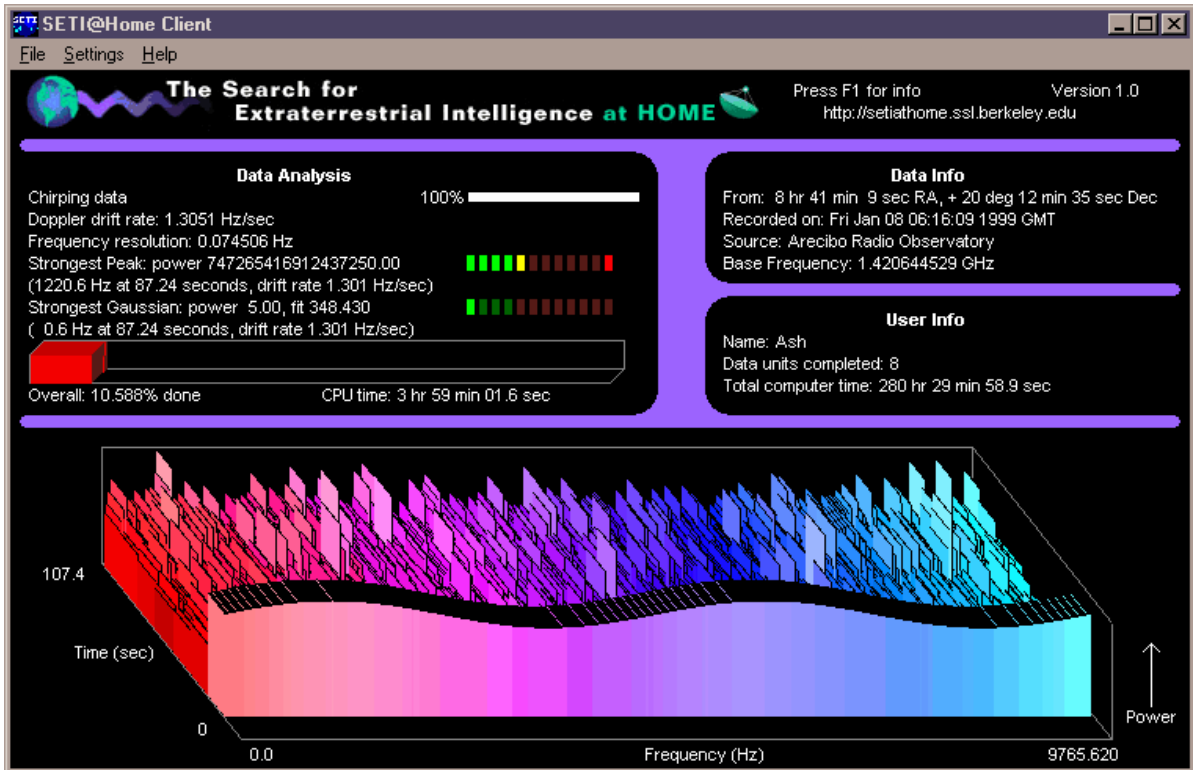


Figure 5 - SETI@home screenshot[15]

Client users can subscribe to one or more projects. A list of URL relating to available projects is maintained at:

http://boinc.berkeley.edu/projects.php

Hidden tags in project web-pages relays the URL of the respective scheduling server of each project, and thus an attachment can be made between clients and BOINC projects.

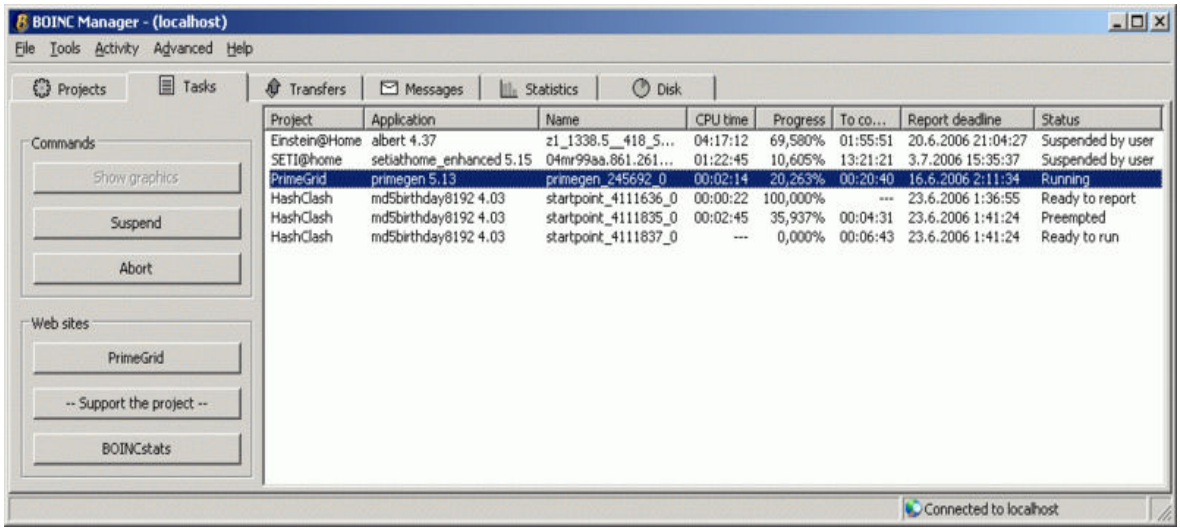


Figure 6 - Account Manager screenshot[16]

BOINC is designed to be implementable on all major platforms, including Windows, Linux, and MacOS. BOINC clients are also available for all the major platforms. It is intended that all BOINC projects and clients can communicate irrespective of the heterogeneity of the platforms on which they are being run.

[Section 2.03 sourced mainly from [17] and [18]]

Section 2.04 Related BOINC projects

Several BOINC projects are in operational in the field of medicine and biology. The one that is most closely related to this FightMalaria@home is MalariaControl.net:

http://www.malariacontrol.net/

The MalariaControl.net project uses stochastic modelling of the clinical epidemiology and natural history of Plasmodium Falciparum Malaria[19]. It models the way malaria spreads in Africa, and the potential impact that new anti-malarial drugs may have on the region[20]. Its ultimate goal is to assess the optimum strategies for the application of mosquito nets and vaccines that are currently in development. It is also hoped that a full range of intervention

and transmission patterns will be discovered[21].

MalariaControl.net is the only BOINC project that is directly addressing the issue of malaria. However, its intention is not to discover a possible drug that might work as a vaccine for malaria, as is the case with this project. Therefore, this project does have a novelty in comparison with other BOINC projects in its ambition of screening large databases of drugs against the malarial genome.

Figure 7 - Screenshot of PrimeGrid web-page

Section 2.05 BOINC on Grids

Related work continues to be done on the theme of integrating BOINC projects with Grids. Researchers at CERN have set up a system in which processing jobs are either sent to a BOINC project, or a Grid Resource Allocation Manager (GRAM) job manager. In fact, two utilities that were developed at CERN have been integrated into the BOINC software platform.

Work is currently being carried out at the University of Maryland to develop a system that integrates Globus, Condor scheduling software, BOINC, and several other software components. The BOINC project is integrated to the Grid as a Globus-addressable resource. The project is called Lattice:

http://lattice.umiacs.umd.edu/

Similarities can be drawn between the Lattice project and the long-term goals for the FightMalaria@home project. Section 12.08 explains that similar technologies and techniques are utilised in both cases.

[Section 2.05 sourced mainly from [22] and [23]]

Article III. Technologies used (I)

The Master Process is implemented in Python 2.4 on (RedHat clone) Scientific Linux 3.07, kernel 2.4.21-40.EL. The built-in unittest[24] module was used for some early test driven development. It was subsequently ported to a grid user interface (UI.testgrid) which runs a similar operating system. A X509 certificate, issued by the college (grid) certification authority was required to access this machine. An instance of MySQL 5.0 was used for the database facility. Public and private keys were generated using ssh-keygen (with the –rsa option). These were placed on various computers which has to be accessed to allow authentication and hence password free access. The *pexpect* library in Python deployed to issue system commands to the operating system, which in turn called the remote machine.

Article IV. Technologies used (II)

The project is run on the Ubuntu 6.1 Linux operating system. The primary technology that is used is the BOINC middleware platform. It provides a library of API functions that are coded in C/C++. To conform with the BOINC toolkit, the majority of the programming development in this project is written in C/C++. An Apache web server is used by BOINC to interact with its clients. This Apache web server is configured specifically for the purpose of this project. BOINC also uses a MySQL database. Most interaction that is performed on the database is done automatically by BOINC; however some manual interaction with the database, using SQL, is required to manage test data.

As most of the code in this project is being developed in C/C++, the make tool and the g++ compiler are used extensively. Finally, files need to be copied remotely in this project; therefore public key authentication is used in conjunction with SCP.

The BOINC framework itself uses some technologies independently, on top of the technologies that are mentioned above. It uses HTTP by way of the Apache web server to both send and receive files and XML data. It also avails of MD5 hashing and public key cryptography to authenticate users and the contents of files.

Article V. Design considerations

Design considerations for this work centred on information gathered during evaluations of the biological data, existing software and the required overall outputs from the work.

Section 5.01 Description of File Types

Automated modelling of the malaria protozoan has lead to the modelling of roughly 400 3D molecular structures which represent these proteins (enzymes or receptors). These 3D models are in human readable ASCII text format. Biological structures modelled in this way, can be used in the area of computer aided drug design and drug-disease matching and screening.

As well as the availability of the protein structures, there are over 20 million freely available structures of drug-like compounds in various databases from academic institutions and pharmaceutical companies. Examples of these are ZINC, MOE and RSCB25.

Section 5.02 Protein Structures

Protein structure files are usually of size 500KB to 2MB. The files contain a number of columns which represent z, y and z coordinates for each atom in the structure. Fig 8 is a graphical visualisation of a protein compound. Throughout this text, proteins will be referred to as proteins and receptors interchangeable.

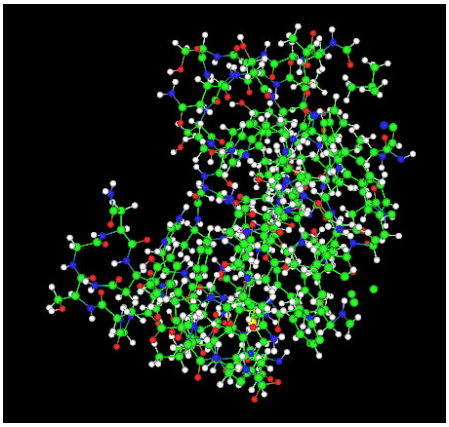


Figure 8 - Graphical representation of a Protein

Section 5.03 *Ligand Structures*

Ligand structures typically have much fewer atoms and consequentially are represented by files of 1 to 3 KB. Fig 9 is a graphical visualisation of a protein compound. Appendix y is an example of a small ligand file from the ZINC database. Protein files have the same format as ligand files, the difference being that they are much longer. Throughout this text, ligands will be referred to as ligands, drugs or drug-like compounds interchangeable.

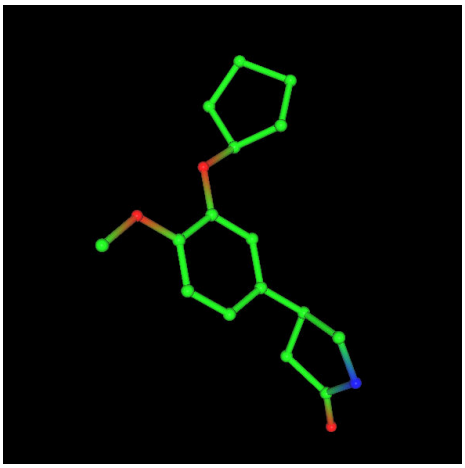


Figure 9 - Graphical representation of a Ligand

Section 5.04 *Docking Energy Between Structures*

All the atoms in these structures have attractions and repulsions towards each other. These attractions are made up of covalent and Van der Waals forces. This attraction energy is measured in Kcal/mol and is known as the docking energy between two structures. In the search for a drug-like compound which will be effective against a malaria protein, a high docking energy is desirable. Throughout this text, docking energy may also be referred to as a score.

Section 5.05 *File Formats for Structures*

In general, molecular structures can be represented in several different ASCII file formats. Across the pharmaceutical and chemical industry, there are several formats. The formats relevant to this project are outlined in Table 1 below.

MDL molecular	File Extension	Comment
SD file	.sdf	Generic, 3D representation
Protein Data Bank	.pdb	Generic
Tripos	.mol2	Generic
Tagged Molecule ASCII	.tma	Native eHiTS ACSII format
Tagged Molecule Binary	.tmb	Native eHiTS binary format

Table 1 - File formats for molecular structures

Drug discovery in a laboratory (i.e. in vitro screening) can be on a trial and error approach and can be non-productive. Due to the large numbers of permutations of proteins and ligands, it is neither economically nor logistically feasibly to carry out in vitro screening for this work. It can however be carried out using computer aided screening software, often called in silico screening.

Section 5.06 SMILES Strings

SMILES™ [26] (simplified molecular input line entry specification) is a comprehensive chemical language in which molecules can be represented by a string ASCII characters. To an expert, SMILES strings are human readable and understandable. In SMILES, atoms, bonds and connectivity of a molecule are depicted in canonical form. They can be generated from most ASCII molecule file formats, and can also be used as a source to generate some ASCII molecule file formats. SMILES strings are used in this project as a contingency in case the ZINC database or any of the other currently freely available drug compound lists becoming unavailable at some point in the future. In that eventuality, SMILES strings will enable a specialist to establish the identity of a compound even if there is no longer a reference available to it in any of the currently available sources.

Section 5.07 Screening Software

The program used for screening in this project is eHiTS (electronic High Throughput Screening). This software was developed by Simbiosys Inc.[27] and they have agreed to allow their program to be used for this academic research work. A somewhat similar program and a candidate for this work is AutoDock[28].

Section 5.08 *Visit to Royal College of Surgeons Ireland.*

The purpose of this visit was to get an initial introduction to eHiTS. Although operated entirely from the command line, trial runs made there indicated that eHiTS was user friendly and easy to run. The program takes one command line in a shell which included file names and various parameters used to govern the speed and accuracy as required. As

the program moves through various stages of operation, progress is indicated at the console, including indicating that the program has ended successfully and the location of the stored files which contain the results. Input files do not need any manual pre-processing before being used by eHiTS.

Section 5.09 eHiTS Installation.

eHiTS is distributed in a single binary file for installation. The installed program comprises of the main docking and scoring application, some utility functions for internal pre-processing, converting and handling different file formats and a set of scripts to automate the overall running of the program.

An evaluation copy can be obtained on request and by agreement from Sysbiosys, Canada. The user will be issued (via. email) with a short text key in a file called ehits.lic. Installation of the program is carried out by running a command on the binary and specifying the path to the installation directory. The binary can be stored in any directory. A typical installation command is,

```
sh eHiTS_6.2_Linux.bin //usr/local/
```

The program will be installed in a directory created within the specified directory called eHiTS_6.2. After installation, the license file must be stored in this directory.

At this point, eHiTS can be run from any directory by specifying this path, for example,

```
//usr/local//ehits.sh -ligand ligands.sdf -receptor
recptor.mol2
```

(a) License Granting

When the program is run for the first time, the program will create a directory called ehits_work in the home directory of the user. It will then attempt to verify the licence granted by contacting the Simbiosys remote license server. It may be necessary to change the proxy settings in the parametes.cfg file within eHiTS to allow this take place. If the transaction with the remote license server is successful, a text file named

`eHiTS.<hostname>.<computer_domain>.<username>.grant`
or... `eHiTS.w02pc2.cs.tcd.ie.lavinp.grant (in this case)`

will be transferred to the client and stored in a directory called license within this directory. This file allows the program to run. eHiTS can now be run for the period of the license (usually one year) and will not need to contact the license server for this time. On all subsequent runs, the presence and validity of this file will be checked the program.

(b) eHiTS Program Operation

The program will use the ehits_work directory to store all its outputs during its operation. A user can specify a different directory for the output files by using the -workdir `<directory_name>` option when running the program. If specifying a directory different to the automatically created ehits_work one, the new directory should be created before starting the program. eHiTS will again need to contact the remote server for using the new directory. If this is not possible, the license folder containing the grant file can be moved there to the new directory.

The two input ASCII files are pre-processed and converted in to a native eHiTS format (tma and tmb). Further files and logs are created at this stage for both the receptor and ligand files. These files are stored in a directory named `preprocess` and can be reused if the ligand or receptor with the same name are processed again. If running multiple

successive screens with different combinations on a single computer, this storage of pre-process data for the structures increases the performance of the program. After pre-processing, docking and scoring part of the program is launched. The output and results from this are also in ASCII format. The output of a docking operation between a receptor and a single ligand will consist of the top thirty docking energies (or scores) measured between the structures. It will also contain the 3-dimensinal position on the receptor at which the top docking energy was calculated. The positional data is known as the *pose* for a particular score. 3D positional pose data for the receptor and ligand for the top score is stored in ASCII SD file format files during the process. This file allows visual examination of the result which gives further insight to the reaction between them. No work was examined in this area for this work. A further script collect the top scores calculated for each of the ligands processed and compiles a summary file named best_scores.txt.

It should be noted that if different ligands are submitted in a file with the same name as used before, the stored pre-process information will be used and hence this may give erroneous results. Using the `-clean` option in the command passed to eHiTS instructs the program to ignore any pre-process information and will generate new data in the pre-process directories for such a run.

(c) Fast and Light Strategy to Screening

From a medical and molecular biology point of view, the pose at which a particular docking energy is measured is important. However, only pose data for positions with a high docking energy are important. Docking energy is a repeatable measurement as was demonstrated by the Simbiosys[32] and by identical scores obtained from repeated trial runs in eHiTS. File comparison using the shell command `cmp` <filename1> <filename2> on the pose and score data files also indicated that the data was identical for repeated screens.

For a screen of 100 ligands against 1 receptor, a total of 11MB of results data is generated. This consists of 30 scores for each individual ligand and a file containing the pose for the best score for each ligand. The file of most interest to this project is best-scores summary file for all the ligands screened. This contains the highest score for each ligand and nothing

else. Given that the vast majority of this pose and individual ligand score data will be for low and uninteresting scores, the strategy of only retrieving and saving the docking energies as summarised in the best_scores.txt file from a screen was decided upon. This was decided in conjunction with Dr. Anthony Chubb in RCSI, Dublin. This approach would generate a list of ligands which had significantly high docking energy scores. This relatively short list (say 400x30 or 12,000 ligand – receptor combinations) could be reworked using eHiTS, AutoDock, GOLD [bib] and other drug screening software programs. At this point, pose data and verification of original results would be obtained using a variety of screening software programs. This approach greatly reduces the storage and data transfer requirements for the entire project. This also simplifies handling and storage of results.

With regard to design purposes, the output of this meeting was as follows:

- ❖ Decision to design around only retrieving the receptor and ligand names and the highest score for that pair.
- ❖ All lower scores and pose data for that pair will be discarded at the computational node.

(d) Comparison of AutoDock and eHiTS

AutoDock consists of two main parts, AutoDock and AutoGrid. AutoDock performs the docking of ligands against a set of grids which describe the receptor or protein. AutoGrid is used to create these grids. This involves some pre-processing on molecule files which must be down before AutoDock can be run. The grid represents the spatial position of each atom in a 3D grid. It can be used for visualisation and other analysis but this type of examinations are not required for this project. The time taken for a screen can vary from 20 minutes to 12 hours per ligand receptor pair. Output is docking energy(s) or score(s) between the compounds, plus the poses (spatial position and orientation of a ligand) at which this docking energy was found. This software has recently become open-source (mid 2007) and is widely used in computer aided drug design. This development happened too late in this work for AutoDock to be considered.

eHiTS uses receptor and ligand files directly and can accommodated several standard ASCII formats. These formats are outlined in Table xxx. eHiTS can screen single or multiple ligands against a receptor. The output is several text file containing scores and poses for a screened ligand. Multiple ligand screens also produce this output plus a best_scores.txt file which contains the best docking energy score for each ligand screened. This information is the most useful in a large drug screen such as in this project. eHiTS also carried out some pre-processing work on files but this is automated within the operation of the program. The program spends a large proportion of the overall time spent on a docking run preparing this receptor grid and graph data for use for the main function of the program. This is illustrated in Table 2 on page 34.

Section 5.10 Estimation of Time Required for challenge

In order to estimate and evaluate the amount of time necessary to make docking runs using eHiTS, a series of tests were run using a variety of receptor and ligand file sizes. These tests were made on the same computer[29] while no other processes were running. CPU usage was close to or at 100% for all of the time taken for these tests. The `-clean` option was used with eHiTS to ensure that no pre-processing data could be accessed.

(a) Relationship of Receptor File size to Docking Time

The time taken to perform a docking run is a function of the number of atoms on or near the surface of the structure. It is independent of the number of atoms in the receptor.[30] For this reason, the time taken is not directly proportional to the file size, but is roughly proportional to the surface area of the structure. As receptor structures are often not spherical in size, the number of atoms and hence file size is only useful as a guide. Table 2 outlines some times measured (in seconds) for pre-processing and docking using five different receptor files. These ranged in size from 105KB to 1.95MB. The proteins are samples from the CCDC Astex Test Set of receptors.[31] The ligands file used contained two

small ligands from the ZINC database. These were of sizes 895B and 1489 respectively, (see appendix 1.0).

Receptor Name	File Size (KB)	Pre-processing time (minutes).	Docking time (minutes).	Total
1aec	105	36	360	396
1a07	416	138	660	798
1pgp	918	1080	1260	2340
1ei1	1530	1440	2040	3480
3gpb	1828	1560	1860	3420
1rt2	1950	4320	1920	6240

Table 2 - Comparison of times for docking of different Receptors against ligands

(b) Relationship of Ligand File size to Docking Time

Of the sample receptors downloaded for this project from CCDC Astex, the average size is 435 KB. A receptor file close to this size was chosen (receptor 1a07) and tested against a variety of ligand file sizes. This test was carried out using a 50 ligand SD file. The individual ligands which made up this set ranged in size form 1855B to 5500B and had an average size of 3402B. This test used pre-processed information for the receptor. The time measured was 6 hours 38 minutes, giving an average of 7.96 minutes per ligand. This time will be used for calculating the overall size of the challenge for this project.

Considering this time, the 400 receptors, and the initial download of 2.02 million ligand files from the ZINC database...

$400 \times 2.02 \times 10^6 = 808 \times 10^6$ ligand and receptor pairs.

$808 \times 10^6 \times 7.96$ minutes $= 6.43 \times 10^9$ CPU minutes or 12,236 CPU years.

Section 5.11 Preparation of Ligand Files from ZINC Database

Data for approximately 2.02 million drug-like compounds were downloaded form the ZINC database via the website of the Department of Pharmaceutical Chemistry at the University of California, San Francisco (UCSF). The files chosen were delivered in large text files (approx 90MB) each containing approx 25,000 individual ligands. Each compound within these large files is delimited by a line containing only the characters $$$$. Using this standard delimiter, a small python script was used to parse individual compounds in to individual files.

It was initially planned to store these individual files in directory on a RAIDS server. Path and file name data could then be stored in configuration file and database respectively. It was intended that files could then be read and written to by code which concatenated this path and file name data. This database table would contain over 2 million relatively short text entries. The motivation for doing this was to minimise the size of the stored data in the database for performance reasons.

During early implementation, testing of this methodology showed that it was not feasible. Access to individual files in a directory with over 100,000 file was many times slower that using a database. For example, when attempting to read a small ASCII file in a directory which contained roughly 100,000 files, search time were over 1 second. This increased to 10 seconds with 500,000 files[32]. This methodology was abandoned in favour of storing all the files (ligands and receptors) in text format in a database. By comparison, storage in a database table using an integer primary key, recovery of a text file of size 1KB from a table with 2,020,000 entries was recorded on at 0.57 seconds (average over three measurements).

(a) Naming scheme for ligands Files

For the purpose of reference and traceability, each of these ligand compounds needed to have an individual name or number which would be unique within this particular drug

screen. It also had to be possible to link the data for the compound to the original source of the file (ZINC, DrugBank, etc). A naming scheme which would be universally applied to all ligand compounds was devised. For simplicity and scalability, a twelve digit number format was chosen, e.g. 000000000001 to 999999999999. This scheme is used for naming individual ligand files during preparatory processing (e.g. 000000000001.sdf). It is inserted in to the text of each file for reference during screening and also as a unique identifier (primary key) when storing the file in a database. It is intended that the twelve characters in the name will always be numeric. Twelve digits provide scope for one less than 1 x 10^{12} ligands. This is many multiples of the quantity currently available in commercial and non commercial databases. This allows for some of the leftmost digits to be used for other identification data.

(b) Existing Naming Scheme in ZINC Database

The clip illustrated here shows the start of file representing a compound from the ZINC database.

```
$$$$
ZINC03831573
  -OEChem-04270608223D
```

The $$$$ delimiter indicates the end of the previous compound as stated earlier. The data on the line immediately after this delimiter (i.e. the first line of the data for a compound) is the unique identifier of this compound in the ZINC database and must not be blank. These numbers assigned in the ZINC database however are not sequential and cannot be used for ordering or sorting compounds or results. This first line could also contain a compound name like Benzene, etc.

(c) SDF Tags

Part of the SD file protocol[33] allows for the use of name tags. This allows a user to store

data in a file which for identification or for use by screening software or any other data a user may wish to associate with a compound. The end of the structural coordinates and atomic information of a compound is marked by 'M END'. Tags are inserted between this marker and the $$$$ delimiter. Multiple tags can be used and there is no end specified to each tag, only the start of the next tag or $$$$ delimiter. Blank lines are allowed for clarity. The following is an example of two tags inserted in a SD file,

```
M  END

>  <example_of_tagname>
INFORMATION ASSOCIATED WITH TAG

>  <second_tagname>
INFORMATION ASSOCIATED WITH SECOND TAG

$$$$
```

Section 5.12 Quantification of Data Storage Requirements

The calculation outlined in section 5.10(b) would produce 808 x 10^6 scores and pose data files. Each of these individual scores is required to be stored as a database table entry.

In order to estimate the storage requirement for a complete set of results receptor, a mock database table was created with the same structure as a planned results table. This was filled with 2.02 x 10^6 simulated results. When examined on the RAIDS server using the OS disk usage reporting utilities, the files associated with this table were 34.7MB. 400 such table are required to store results for the proposed number of receptors, requiring about 15GB for results. This storage requirement will increase directly in proportion to the number of ligands screened.

For an overall screen of 20 x 10^6 ligands, a storage capacity of roughly 127GB is required

for results alone. Further capacity is needed for preparation and storage of the ligand and block files. This requirement is outlined in Table 3 below.

Description	**Table name**		**For 2.02 x 10^6 ligands in ZINC**	**For $20x10^6$ ligands**
Table containing protein/receptor files	Tbl_prot_list		10MB	10MB
Table containing individual ligand files	tbl_ligfiles		7.8GB	71GB
Table containing block files	Tbl_b100files		7.8GB	71GB
Tables for each protein jobs (400 off)	tbl_<prot>_jobs	(256KB x 400)	100MB	910MB
Table for each protein results (400 off)	tbl_<prot>_results	(35MB x 400)	14GB	127GB
		Approximate Total	**30GB**	**270GB**

Table 3 - Storage requirements

Section 5.13 Database Design Consideration

Although the expected storage requirement for the database in this work is approximately 270GB, the tables which contain the returned scores for each ligand could potentially have up to $20x10^6$ entries.

There is no formal standard definition for the term Very Large Database (VLDB), the perception of what constitutes a VLDB continues to grow. A one terabyte database would normally be considered to be a VLDB this term is sometimes used to describe databases which occupy upwards of 1TB of magnetic storage (excluding replication using RAIDS)[34]. Examples are available of database tables with up to 50 billion rows[35]. Such databases are used for decision making support in large retail organisations.

The structure of each of the jobs and results table is the same. Using Third Normal Form (3NF) rules [36], it would have been possible to construct a single table and include a field to indicate which receptor scores belong to. This would have simplified the database design and reduced the number of tables needed to store the data. However, using such a table

would mean all the results for all proteins would be stored in one large table (800 million rows for results from the ZINC database alone). A schema where the jobs and results for each receptor are stored in individual tables was chosen.

The database used for this project is a dedicated instance of MySQL 5.0 and is stored on a raids server within the CS department. This machine has 4.5TB of storage and is intended for use as a backup server and storage repository.

Article VI. Overview of BOINC

BOINC is a software platform for developing projects for the purpose of Volunteer and Desktop Grid Computing. API functions are made available in the C/C++ programming language with which to develop a BOINC application. Various architectural components are present in the BOINC platform, and are discussed in the next section, 6.01. The concept behind BOINC is to divide up the total processing work to be done into work-units that the BOINC clients can then process, resulting in meaningful data on the completion of each work-unit.

Section 6.01 Architecture of a BOINC server

Along with the BOINC project tools and libraries themselves, a number of architectural components are required to create a fully-functioning BOINC server. The complete list of components is as follows:

- BOINC tools and libraries
- MySQL database
- Apache web server
- BOINC project

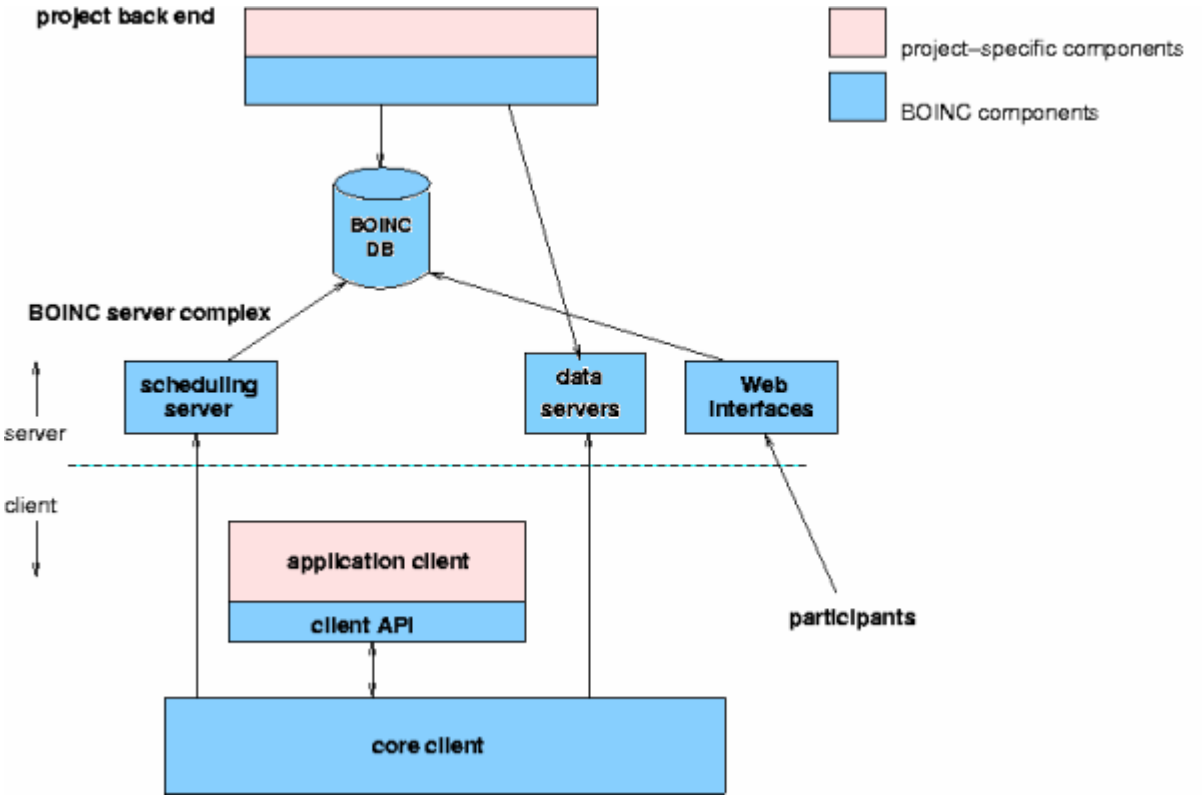


Figure 10 - BOINC architecture[37]

(a) BOINC tools and libraries

The tools and libraries of BOINC can be acquired by downloading the BOINC software. The developer of a BOINC project has two primary uses for the BOINC software, they are:

- To the install the BOINC shared libraries onto the system of the server
- Calling the make_project script

Having the BOINC libraries installed on to the system means that the BOINC API functions can be called from a BOINC project that is set up at any location on the server machine. The make_project script is called to set up a BOINC project at a user-defined location, complete with the necessary directory structure, scripts, binaries, server code, and web-functionality that it needs to operate.

A basic call to the make_project script would be as follows:

```
$ make_project –project_root [project_path] –db_user [user] –db_passwd [passwd]
project_name Project_long_name
```

This call creates a BOINC project (see section 6.02) in the directory specified at –project_root, associates the project with an instance of MySQL, sets up the tables required to maintain the project (see section 6.02(b)), and gives the project a naming scheme.

(b) MySQL database

A MySQL database is required to operate a BOINC server. Information relating to volunteer nodes, together with data processing details, is stored in the database. See section 6.02(b) for details.

(c) Apache Web Server

All interactions between the BOINC server and volunteer nodes are conducted using CGI over HTTP by way of an Apache Web Server. Scheduling requests and responses, as well as file uploads and downloads, need to take place to ensure that sufficient information is passed between server and client to carry out the biological screening and subsequent processing.

The CGI scripts that are present in the cgi/ directory (see section 6.02(a)) in the BOINC project root directory are served from the Web Server. These scripts essentially schedule out work for processing, and collect the results that are returned.

The Web Server also provides a means for the BOINC server to be administrated remotely by way of a web-page front-end. It also enables admin-customizable web-page to be displayed to users so that they can find out about the project and its goals. Both of these web-page options are achieved by using and tailoring PHP scripts that are provided with the

BOINC software. These PHP scripts are present in the html/ directory (see section 6.02(a)) of the BOINC project root directory.

The Web Server can be configured by altering the httpd.conf file in the Apache installation hierarchy. In this manner, both the URLs of the html/ directory and the cgi-bin/ directory can be specified, and therefore served, by the Web Server to potential volunteer users. The IP address and port number is also specified. The PHP module is also enabled in Apache so that the web interface can be used for the (remote) configuration of the project.

Section 6.02 BOINC project

The BOINC project consists of three components:

- Directory structure
- Database
- Configuration file

(a) Directory structure

The directory structure is laid out as follows:

```
PROJECT/
    apps/
    bin/
    cgi-bin/
    log_HOSTNAME/
    pid_HOSTNAME/
    download/
    html/
        inc/
        ops/
```

```
        project/
        stats/
        user/
        user_profile/
    keys/
    templates/
    upload/
```

Apps

This directory contains all applications that are to be released with the project, as well as all the versions that fall under each application.

Bin

This directory contains binary files which enable the user to manipulate the project. It also contains the executables of user-defined and BOINC daemons.

Cgi-bin

This directory contains three CGI scripts which are to be hosted by the Apache Web Server.

Download

This directory contains all the user-created files that can be associated with work-units, and can therefore be downloaded by a client user node.

Html

This directory contains many files that provide the web-functionality for the project to be administered remotely through a web-page that is hosted by Apache. It also provides a project-admin customizable web-page to be viewed by volunteer users so that they can find out about the project.

Keys

The purpose of this directory is to store the public and private keys that are used as a means to sign code, files, and executables, so that a user can be assured as to the identity of the

sender of any potentially malicious data.

Templates

This directory contains the two generic templates that relate to either the input file(s) of each work-unit, or the output file(s) of each result.

Upload

This directory contains all the results files that have been sent back from the processing of work-units on the volunteer nodes.

(b) Database

The database schema is automatically generated by the make_project script. The database information concerns most aspects of the project including work-units, results, applications, users etc. Insertions and deletions to and from the database are carried out automatically by BOINC functions, and do not require specific input from a project administrator. The tables are seen in the database as follows:

- Platform
- App
- App_version
- User
- Host
- Work-unit
- Result

Platform

This table contains the compilation targets of the core client and/or applications.

App

This table contains information about applications; a project can have many applications. The core client is treated as an application; its name is 'core_client'.

App_version

This table describes the versions of each application. Each record includes a URL for downloading the executable, and the MD5 checksum of the executable.

User

This table describes users, including their email address, name, web password, and authenticator.

Host

This table describes the volunteer hosts.

Work-unit

This table describes work-units. The input file descriptions are stored in an XML document in a blob field. Includes counts of the number of results linked to this work-unit, and the numbers that have been sent, that have succeeded, and that have failed.

Result

This table describes results. Each result includes a 'state' that indicates whether it has been returned, validated etc. It stores a number of items relevant only after the result has been returned: CPU time, exit status, and validation status.

[Database table descriptions sourced mainly from [38]]

(c) Configuration file

The project configuration is described by the file, config.xml, that is located in the project root directory. It can be configured to change the options of the project, to change the number and operation of the project daemons, and to specify periodic tasks that are to be carried out to monitor the operation of the project. More details are provided in section 6.03(b).

Section 6.03 Further tools and configuration files

(a) Project.xml

The project.xml is located in the project root directory, and takes the following form:

```
<boinc>
    <app>
       <name>setiathome</name>
       <user_friendly_name>SETI@home</user_friendly_name>
       [ <min_version>N</min_version> ]
       [  <homogeneous_redundancy>0|1</homogeneous_redundancy>
]
       [ <weight>X</weight> ]
       [ <beta>1</beta> ]
    </app>
    ...
    <platform>
       <name>anonymous</name>
       <user_friendly_name>anonymous</user_friendly_name>
    </platform>
    <platform>
       <name>i686-pc-linux-gnu</name>
       <user_friendly_name>Linux/x86</user_friendly_name>
    </platform>
    <platform>
       <name>windows_intelx86</name>
       <user_friendly_name>Windows/x86</user_friendly_name>
    </platform>
    <platform>
       <name>powerpc-apple-darwin</name>
       <user_friendly_name>Mac OS X</user_friendly_name>
    </platform>
    ...
```

```
</boinc>
```

Each application that the BOINC project operates is described within each set of <app> tags. The purpose of these tags is to assign, among other things, a name and a user-friendly name to the application. The <user_friendly_name> is how the users that operate the BOINC clients refer to the application, and the <name> is how the application is referred to during the development process of the BOINC project.

One or more <platform> tags can be added to detail the platforms under which each application can potentially run on the BOINC clients.

(b) Config.xml

The config.xml configuration file for the project is located in the project root directory, and takes the following form:

```
<boinc>
  <config>
    [ configuration options ]
  </config>
  <daemons>
    [ list of daemons ]
  </daemons>
  <tasks>
    [ list of periodic tasks ]
  </tasks>
</boinc>
```

There are many options that can be specified in the <config> tags. They fall under the following topics:

- Scheduling options and parameters
- Client control
- Server logging
- Credit
- File deletion policy
- Server status page options
- Website features
- Hosts, directories, and URLs

The primary topic of interest for the development of a BOINC project is the last topic – Hosts, directories, and URLs. Its options help to specify all of the important locations, and server resource locators for the BOINC project. They are as follows:

```
<master_url> URL </master_url>
<long_name> name </long_name>
<host> project.hostname.ip </host>
<db_name> databasename </db_name>
<db_host> database.host.ip </db_host>
<db_user> database_user_name </db_user>
<db_passwd> database_password </db_passwd>
<shmem_key> shared_memory_key </shmem_key>
<download_url> http://A/URL </download_url>
<download_dir> /path/to/directory </download_dir>
<uldl_dir_fanout> N </uldl_dir_fanout>
<upload_url> http://A/URL </upload_url>
<upload_dir> /path/to/directory </upload_dir>
<cgi_url> http://A/URL </cgi_url>
<log_dir> /path/to/directory </log_dir>
```

Most of these are set automatically when the make_project script (see section 6.01(a)) is invoked. However, they can be re-configured to suit the needs of the project.

BOINC daemons are discussed in section 6.10. Only the daemons that are specified to run in the configuration file tag <daemons> are actually started when the BOINC project is initiated. Daemons can be optionally turned off to suit the needs of the project.

Both project-dependent and project-independent tasks can be specified in the <tasks> tag. The tasks that are specified are run periodically in accordance with the configuration of the cronjob UNIX daemon. An example for what a task could be used would be generating statistics on the performance of the work-units.

(c) Upload/Download directory structure

A hierarchical scheme is employed in both the upload and download directories to prevent needless CPU time being spent searching through potentially thousands of upload or download files. The default fan-out value is set to 1024. This means that a maximum of 1024 sub-directories are created and are named hexadecimally, with names ranging from 0 to 3ff. In order to determine which files go under which directories, the name of a file is hashed. The file is then placed in a directory, the name of which corresponds to its own hash value.

A C/C++ API function is provided to resolve the fan-out directory that corresponds to a certain file. It is of the form:

```
int dir_hier_path(
     const char* filename,
     const char* root,
     int fanout,
     char* path,
```

```
    bool create
);
```

The char pointer `path` is passed into this function. When the function returns, `path` contains the absolute path (including the fan-out directory) in which the download file, under the name `filename`, should be placed. This function is mainly used for preparing input files for the download/ directory.

Separate back-end functions are used in project-specific daemon code to resolve the path of uploaded files in the upload/ directory. These functions take the form:

```
int get_output_file_path(RESULT const&, std::string&);

int get_output_file_paths(RESULT const&, vector<string>&);
```

The first function is used to resolve the path of a canonical result of a work-unit. The RESULT class instance represents the canonical result, and the path to the uploaded file is returned in the string data-type that is passed to the function.

The second function is used to resolve the paths that correspond to all of the files that were returned for each result of a work-unit. In this case, a vector of strings is returned containing the path to each of the uploaded files.

(d) Schedulers.txt

The BOINC client can be attached to a BOINC project by specifying the master URL that is defined in the config.xml configuration file (see section 6.03(b)). A <scheduler> tag is hidden in the HTML code of the master URL web-page. The URL for the primary CGI script from the hosted cgi/ directory (see section 6.02(a)) of the project is contained in the scheduler tag. In this manner, a BOINC client may connect with and receive work from the server.

Section 6.04 Security

Two sets of asymmetric keys are needed to deal with the two major security issues that are associated with BOINC. The first issue is the dissemination of malicious executables. The second is denial of service attacks to the BOINC server by uploading abnormally large files. The keys can be generated using a tool that is provided with BOINC called crypt_prog.

A set of keys is generated to prevent the possibility of hackers spreading malicious executables among the clients, should they happen to hack the BOINC server. This set of keys is referred to as the code-signing key pair. The private key is stored in a network-isolated location away from the BOINC server. The public key is distributed to all clients. Before an executable is released to the clients, it has to be accompanied with a signature file. The signature file is a digital signature of the executable signed with the private key. As an option, all data files that are associated with the executable can also be signed. Each executable that the client downloads can therefore be authenticated as officially coming from the BOINC project.

The second set of project keys is known as the upload authentication key pair. Its purpose is to prevent the success of denial of service attacks on the BOINC server. A result file description, that is digitally signed, is sent to each client. When a result file is to be uploaded by a client, it also uploads the digitally-signed result file description. The signed file description can then be verified on the server, and it can then be known that the size of the result file does not exceed pre-defined limits.

Section 6.05 Applications

An application is a program that is designed to carry out a certain processing goal on the BOINC Client. A BOINC project can operate more than one application. Versions of each

application are released to run on different architectural platforms. Furthermore, each application version has a version number that is continuously updated as changes to the code take place to suit the dynamic requirements of the application itself.

(a) Application API functions

Various API functions are provided by BOINC to enable the BOINC client to monitor the progress and status of the application as it executes, and to enable the application code to interact with its environment and locate the data files that it needs to process.

The initialization and termination functions take the form:

```
int boinc_init();

int boinc_finish(int status);
```

All of the other BOINC API calls have to be called between the interval of initialization and termination, with exception of the diagnostics function that takes the form:

```
int boinc_init_diagnostics(int flags)
```

This function initializes various diagnostics and ensures that error information is channelled appropriately. A function is also provided to resolve logical to physical filename mappings (see section 6.07). It is of the form:

```
int boinc_resolve_filename
     (char *logical_name, char *physical_name, int len);
```

Additional functions are provided, such as an I/O wrapper of the form:

```
boinc_fopen(char* path, char* mode);
```

This function facilitates the opening of a file using the physical name that is returned from the boinc_resolve_function(…) above. It differs from normal C/C++ file functions due to the fact that it deals with platform-specific issues that occur when a BOINC project is designated to run on more than one platform.

(b) Adding applications and platforms

The xadd binary file is located in the PROJECT/bin/ directory. It parses the project.xml file and updates the database with any applications or platforms that may have been added to the file and that are not already present in the database. It also creates sub-folders under PROJECT/apps/ for each application, where the versions relating to each platform can be stored.

(c) Updating the version

Each platform version takes the form of a C/C++ executable. The name and directory under which the executable is saved determines the respective platform and application with which it is associated.

```
PROJECT/
     apps/
          MyApplication/
               myapplication_1.1_i686-pc-linux-gnu.exe/
                    myapplication_1.1_i686-pc-linux-gnu.exe
                    data_file.txt
                    license_file.txt
```

The directory structure above gives an example of an executable called myapplication_1.1_i686-pc-linux-gnu.exe being associated with an application called MyApplication. Implicit in the title of the executable is the version number, 1.1, and the platform, i686-pc-linux-gnu. In order to associate extra data files with an executable, a

directory, named after the executable, is created, under which the executable itself and all relevant files are placed. This means that when a BOINC client downloads the executable, it also downloads the extra files. These files are used by the executable as it executes on the BOINC client.

When the appropriate executables and files have been added under the correct directories, the update_versions binary file, located in PROJECT/bin/, is called to add the new data to the database.

Section 6.06 Generating work

Each computation to be carried out on the volunteer nodes is encapsulated in a work-unit. Typically thousands of work-units are prepared in advance to provide a constant supply of work to the volunteer nodes. Various binary files and C/C++ API functions are provided with the BOINC software to prepare files for download, and to generate the actual work-units. These files and functions are discussed in the following sections.

(a) Preparing input files

In order for input files to be associated with a work-unit, not only do they have to be present in the download/ directory, but they also have to be placed in a sub-directory appropriate to the hierarchical fan-out scheme. See section 6.03(c) for details.

Both a binary file (located in the bin/ directory), and a C/C++ API function are provided to move files into their correct sub-directory in the download/ directory. The call to the binary function is of the form:

```
cp test_files/12ja04aa `bin/dir_hier_path 12ja04aa`
```

This call places the file 12ja04aa, that is located in the directory test_files/, into the

download/ directory under the appropriate sub-directory.

The C/C++ API function can be seen in section 6.03(c). This function call has to be made in conjunction with project-specific code to actually move the file to the upload/ directory.

(b) Creating a work-unit

When a work-unit is created, a unit of XML data, and related information, are inserted into the *workunit* table of the database. All of the information relating to a work-unit is stored there, including the following data:

- The application version to which it is related
- The input file(s) which relate to it
- The work-unit template which describes the input file(s)
- The command line arguments that will be passed to the application executable of the work-unit
- The result template XML file to which its future results are related

Tools are provided by BOINC to create work-units. All of the data relating to a work-unit can be specified to these tools. Both a binary file and a C/C++ API function are provided to create work-units. The call to the binary file is of the form:

```
create_work -appname [name] -wu_name [name] -wu_template [filename] -results_template [filename] infile_1 ... infile_n
```

Additional arguments include `-command_line`, and `-additional_xml`, which mean that this call can be as powerful as the API function; however using the API gives more versatility when used in a project-specific program.

The C/C++ API function is of the form:

```
int create_work(
   DB_WORKUNIT&,
    const char* wu_template,
    const char* result_template_filename,
    const char* result_template_filepath,
    const char** infiles,
    int ninfiles
    SCHED_CONFIG&,
    const char* command_line = NULL,
    const char* additional_xml = NULL
);
```

For each work-unit there has to be one or more 'results'. There are tables in the database for both work-units and results (see section 6.02(b)). Each result corresponds to the data that is returned from a BOINC client after the processing of a work-unit. The Transistioner daemon (see section 6.10(b)) creates one or more results each time a work-unit it created. The number of results that are created for each work-unit depends on the factor of redundancy (see section 6.07(a)) that is being used.

Section 6.07 XML Templates

Two XML template files need to be written to describe both the files that are sent out to a volunteer node with a work-unit, and the files that are received back from a volunteer node as results. The primary purpose of the templates is to specify the physical to logical naming that is to be used on the BOINC client. This means that irrespective of the name of an input file, files can still repeatedly be opened and created on the client, and returned to the BOINC server under the same names for different work-units.

Two sets of XML tags are used in common in the work-unit and the result templates. They are the `<file_info>` and the `<file_ref>` tags.

The information enclosed in the <file_info> tag describes the properties and status of the file when it is on the client, and on what conditions the file should be uploaded or downloaded.

Each <file_info> tag is related to a <file_ref> tag with the used of a <number> sub-tag that is common to both. The <file_ref> describes how the file is referred to by an application executable during the lifetime of a work-unit through the use of logical naming. When the file is downloaded by the BOINC client, it is saved under the logical name.

(a) Work-unit template

The work-unit template is of the form:

```
<file_info>
    <number>0</number>
   [ <sticky/>, other attributes]
</file_info>
[ ... ]
<workunit>
   <file_ref>
      <file_number>0</file_number>
      <open_name>NAME</open_name>
   </file_ref>
   [ ... ]
   [ <command_line>-flags xyz</command_line> ]
   [ <rsc_fpops_est>x</rsc_fpops_est> ]
   [ <rsc_fpops_bound>x</rsc_fpops_bound> ]
   [ <rsc_memory_bound>x</rsc_memory_bound> ]
   [ <rsc_disk_bound>x</rsc_disk_bound> ]
   [ <delay_bound>x</delay_bound> ]
   [ <min_quorum>x</min_quorum> ]
```

```
    [ <target_nresults>x</target_nresults> ]
    [ <max_error_results>x</max_error_results> ]
    [ <max_total_results>x</max_total_results> ]
    [ <max_success_results>x</max_success_results> ]
    [ <credit>X</credit> ]
</workunit>
```

The important issue of redundancy is addressed in the work-unit template. Redundancy is used as a fail-safe to account for spurious results that may be returned from clients that either run on different platforms, or that are operated maliciously. The factor of redundancy can be set in the work-unit template using the <target_nresults> tag. The factor of redundancy dictates how many times the work-unit is to be sent out to different clients.

A consensus has to be reached in order for one particular result to be deemed valid. A consensus is reached when a certain number of results for the same work-unit are deemed to be the same within certain limits (see section 6.10(f)). The number of results that are needed to reach a consensus is defined in the <min_quorum> tag. The result with which the consensus is reached is known as the canonical result.

(b) Result template

The result template is of the form:

```
<file_info>
   <name><OUTFILE_0/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>32768</max_nbytes>
    <url><UPLOAD_URL/></url>
</file_info>
<result>
    <file_ref>
```

```
        <file_name><OUTFILE_0/></file_name>
        <open_name>result.sah</open_name>
    </file_ref>
</result>
```

The primary purpose of the result template is to describe the results files that are to be uploaded to the BOINC server when a work-unit has completed. In the template above, the generic tag <OUTFILE_0/> is placed in between the <name> and <file_name> elements. When the result file is initially being created, its name is a variation of this tag and of the work-unit name. For example, the sole result file for a work-unit called 'my_work-unit', that has a redundancy factor of 1, would be:

```
my_work-unit_0_0
```

The first zero indicates that this file is the first (and this case the only) file that is to be uploaded. The second zero represents the order that this work-unit occupies in the sequence of redundant work-unit jobs (in this case it is the only one).

The <url> tag is used to specify the URL (replaces <UPLOAD_URL/>) of the CGI script on the server that accepts file uploads from BOINC clients.

Section 6.08 BOINC web-page

A project needs a web-page to inform potential users about the project, and to maintain the interest of existing users. The web-page URL appears among those other BOINC projects on the BOINC web-site (see figure 4). A generic interactive template is provided with BOINC to configure the web-page. The primary resource is the index.html page in the html/ directory in the project root directory. It can be edited, along with the white.css style sheet, to alter the information and appearance of the web-page. More dynamic control of the web-page behaviour can be yielded by configuring the project.inc file. A number of values have to be set in this file that will automatically be displayed on the web-page. Several PHP functions also need to be customized to add the desired behaviour to the web-

page. Additionally, some values can be set to enable PHPMailer. This means that emails can be sent to users under certain circumstances.

Section 6.09 Screensaver graphics

It is recommended that screensaver graphics be implemented with a BOINC project. These graphics give the user a dynamic representation of the progress of computations that are running on their machine, and move to maintain user interest in the project. A graphics API is provided with BOINC. It enables graphics code to either be integrated with the main application, or implemented as a separate program. Developers are encouraged to develop their graphics code in OpenGL as it is portable to all OS platforms. Several rendering and input-handling functions are called periodically by BOINC and need to be customised by the graphics developers to correctly integrate the graphics with the BOINC framework.

Section 6.10 BOINC daemons

A set of daemons are provided with the BOINC project for generating and handling work. They run continuously, and are either supplied by BOINC as being project-independent, or implemented by the operator of a BOINC project. In all, it is suggested that seven daemons are run to maintain a fully functioning BOINC project. Four of these daemons are project-independent, and they include:

- Feeder
- Transitioner
- File Deleter
- Database Purger

The three remaining daemons are project-specific, and they include:

- Work Generator
- Validator
- Assimilator

(a) Feeder

The Feeder creates a shared-memory segment that it used to pass database records to the CGI scheduler processes (see section 6.12(b)). These database records include information on applications, application versions, and work-units.

(b) Transitioner

The Transitioner is responsible for changing the status of both work-units and results over the course of their lifetime. It also generates database 'results' that correspond to work-units that have just been created.

(c) File Deleter

The File Deleter deletes input and output files when they are no longer needed. This is done so as to prevent a back-log of out of date files clogging the machine on which the server resides.

(d) Database Purger

The Database Purger removes old records from the database that correspond to work-units that are no longer needed. This maintains the database at a reasonable size even when the project has been running for a considerable length of time.

(e) Work Generator

The Work Generator creates work-units using either the BOINC library API functions, or the BOINC binary commands. It creates new work-units depending on the status, or rate of completion of outstanding work-units.

(f) Validator

The purpose of the Validator is to compare the redundant results that are returned for a work-unit and to assess whether a canonical result can be selected that corresponds to the correct output for the work-unit. Functionality is needed to specify to which extent results can differ before they are deemed to be dissimilar. Results can vary due to the different instruction sets that can found on different hardware, or the different floating point specifications that can be found with different software.

As the Validator is invoked by the Transitioner, the Transitioner daemon can change the status of the results once the Validator has selected a canonical result. They can then be processed by the Assimilator daemon.

(g) Assimilator

The Assimilator handles completed work-units, whether they have returned with a canonical result, or have returned in error. Handling these work-units could entail creating more work or recording results into a database.

Section 6.11 BOINC Client

The BOINC client is a BOINC-specific software component. Volunteers can download this software from http://boinc.berkeley.edu. Using the software, the user can subscribe to many

projects of their choosing. To subscribe to any one project, the user has to specify the host URL of the project, and choose a user-name and password. Having successfully attached to a project, information relating to the user will be stored in the project database (see section 6.02(b)), including any work-units or results in which they may be involved.

(a) Directory structure

There are two directories in the BOINC client that are of interest from the perspective of a BOINC project developer. They are as follows:

- The projects directory
- The slots directory

The projects directory contains a sub-directory for each project to which a user has subscribed. All the work-unit and application-related files for a project are downloaded into a directory relating to a project. Results files are staged here so that they can be uploaded to the BOINC server.

Project specific executables, as well as application specific files, and work-unit files, all of which are downloaded from the BOINC server, are loaded into the slots directory before a work-unit is processed. Each executing work-unit is given its own particular slot directory by the BOINC client. The slots directory can be seen as the working directory when an application executable is actually executing. Any system calls that are made to work-unit-specific files or executables by the application executable can be referenced easily as all relevant files are automatically present in the same directory.

(b) Work-unit template

See section 6.07(a) for details on work-unit templates. The BOINC client uses the work-unit template to ascertain how input files are to be received for a work-unit and what attributes they have. The most important feature that the work-unit template lends to a file

is a logical name that is assigned to the input file irrespective of what the physical name might be. This means that application-specific code can be generic, and refer to the same logical name to access the data of the different files that are associated with each separate work-unit.

(c) Results template

See section 6.07(b) for details on result templates. The BOINC client uses the results template to retrieve the output files that need to be sent to the BOINC server. In the same vein as the work-unit template, a logical to physical naming scheme is used. The application executable generates the same number of files under the same names on completion of each work-unit. Multiple files under the same name cannot be sent back to the BOINC server. Therefore, a different logical name is given to each file that is generated per work-unit, and is given a name that is a variation of the work-unit name. The files can then be sent to and are readily identifiable at the BOINC server.

(d) Doing work

The BOINC client periodically polls the server for work-units. When a work-unit is first sent to the client, the client automatically downloads all of the files relating to the application with which the work-unit is associated, including the application executable, and stores them in the projects directory. Each work-unit may also have files related to it, and each of the files is downloaded along with their respective work-units. To process each work-unit, the client calls the application executable. Result files are then uploaded to the server when the execution has finished.

Should the client receive one or more work-units for the same application version, then it does not need to re-download the executable or the associated files as they are maintained in the projects directory.

Section 6.12 Life cycle of a work-unit

Typically, the collective amount of computation that has to be done for a BOINC project is huge. This work has to be divided up into units small enough to be processed on a BOINC client in a reasonably short amount of time (six hours is a feasible goal). These units are referred to as work-units. A work-unit can be seen to go through a cycle. This cycle is described in the following section. Any references that are made to architectural components in this section are done so with regard to figure 10.

(a) Creating a work-unit

Firstly, before a work-unit can be created, an executable must be developed and made known to the BOINC server. When the work-unit is created, it is associated with this executable. The executable dictates the processing that takes place on the BOINC client. A work-unit also has to be linked with one or more input files. The work-unit can be seen as the goal of processing the data that is contained in the input file(s) with which it is linked. Furthermore, two XML templates are linked with the work-unit, as well as a list of command line arguments that are to be passed to the executable. The first XML template describes how the BOINC client should interpret the input file(s) of the work-unit. The second XML template indicates to the BOINC client how to find and interpret the result file(s) that is/are produced when the work-unit is processed.

(b) Downloading a work-unit

To receive a work-unit, a BOINC client first has to contact the Web Interface by way of a URL. The Web Interface refers the BOINC client to the URL of the Scheduling Server. Data relating to the work-unit is passed to the BOINC client by the Scheduling Server, along with the URL of the Data Server. The Scheduling Server refers to the BOINC DB to discover about the work-unit. The BOINC client can download all the files relating to the work-unit from the Data Server. As a work-unit is associated with an executable, it needs to be downloaded, and is downloaded first. Should subsequent work-units be associated with

the same executable, it does not need to be downloaded again. The files and information relating to the work-unit can then be downloaded. They include the input file(s), the XML templates, and the command line arguments.

(c) Processing a work-unit

The BOINC client uses the XML template that describes the input file(s) to prepare the file(s) for the executable. The BOINC client passes the command line arguments to, and then invokes, the executable, whose code processes the data that is contained in the input file(s). The processing ultimately produces one or more result file(s).

(d) Uploading results

Using the XML template that describes the result file(s), the BOINC client can locate and identify the result file(s) that need(s) to be uploaded to the BOINC server. The relevant result file(s) can then be uploaded to the Data Server.

(e) Back-end processing

Once the results are uploaded onto the server, the back-end daemons can then step in to analyse the results and take actions depending on the results. The details of the analysis that is done and actions that are taken are project-specific. Once the results are processed successfully, the work-unit can then be marked as complete.

Article VII. Implementation (I)

This section deals with the implementation of the Master Process and the preparation of data for use by that process. It also outlines the installation of both the standard and third party libraries used in the Master Process. Early implementation was carried out using the Eclipse framework[39] and a Python plug-in. Scientific Linux 3.07 was used for this part. Although some libraries were installed using build in update mechanisms in the operating system (rpm[40], apt, etc.), later and final installation were made by using tar-files to facilitate the 'single directory' installation. This method gives the user more control over the destination of the installed libraries.

Section 7.01 Python 2.4 setup for Master Process

In order to allow this code and installation to be easily moved and installed on several computers, all the required software and packages used in the Master Process are stored in one directory below `//home/username/` called `fightmalaria/`. Using a separate installation tree makes portability between computers much easier.

(a) Python 2.4

The Python-2.4 binary and associated files were installed in a directory tree of the form `/usr/local/bin`, similar to how it would be installed at the top of the file system. Although the current version of Python deployed on the UI test grid computer is Python-2.2.3, for reasons extensibility in the area of web services in this project Python-2.4 is used. Using a separate directory tree for the installation makes this possible. A short initialising shell script was written and stored in the `fightmalaria` directory. Before executing the program, this script file is run at the command line by typing `source setup.sh`.

For initialising python, this shell script is as follows...

```
export PATH=`pwd`/usr/local/bin:$PATH
export PYTHONPATH
```

This sets the environment variable `PATH` to look for the python2.4 binary in the folder specified as opposed to `//usr/local/bin` at the head of the file system. Such settings would usually be placed in the `.bashrc`. Using such a `setup.sh` file allows the `fightmalaria` folder to be moved without having to make any changes to the `.bashrc` file on the destination computer. Using this type of installation means however that all future packages installed for use in folder will have to be installed in the same manner as what is already installed.

(b) OEChem Software Licensing and Installation

A written application stating intentions and scope of use, nondisclosure agreement, etc. must be submitted to Eyesopen Scientific Software in order to obtain an academic license. Details of how to apply for this license are outlined at the Eyesopen website[41].

Eyesopen will then issue a successful candidate with a licence in the form of a text file. Using this license as required at website interface, a tar file is downloaded OpenEye-python-1.5.1-1-centos-3.6-g++3.2-i586-python2.4.tar.gz from www.eyesopen.com download folder.

The libraries are installed by moving the tar file to the `usr/local/` directory. Unzipping the file here creates a directory called openeye and creates a tree structure as follows:

`/usr/local/openeye/wrappers/python/openeye/`
or
`~/fightmalaria/usr/local/openeye/wrappers/python/openeye`, in this case.

For the installation used in this project, the following variables were appended to the `setup.sh`. file.

```
PYTHONPATH=`pwd`/usr/local/openeye/wrappers/python/
export LD_LIBRARY_PATH
```

Instructions for installation are outlined in the bundled documents with the downloaded file, see `/usr/local/openeye/docs/pdf/pyprog.pdf` after installation.

(c) Other packages Installed

The following list of tar files are also installed in the `fightmalaria` directory. These can be installed by running expanding the tar file and running `python setup.py` in the appropriate directory.

MySQL-python-1.2.0.tar.gz
mysql-4.0.27.tar.gz
pexpect-2.1.tar.gz

Section 7.02 Inserting Naming Scheme Data

Using tags allows the number assigned to a compound to be inserted for use when screening. For this, the twelve digit number was appended to the end of the ZINC number. This gives the format `ZINC03831573_000000000001` and links the compound to the original number or name which came from the source database (e.g. ZINC in this case). Using the original number from the source database in the tag meant that the first line of the compound did not have to be changed. This was done so that files used in this project would retain their original name on the first line and so could be used elsewhere. The tag name chosen for this work was `fm_tagname`. An example of an inserted tag is shown here...

```
6 15  1  0  0  0  0
M  CHG  4  12  -1  13  -1  14  -1  15  -1
M  END

>  <fm_tagname>
ZINC03831573_000000000001

$$$$
```

Section 7.03 Insertion of SMILES Strings

In this project, SMILES strings were generated from the ligand SD files. The following is an example of a SMILES string, `CC(=O)NCCCCCC(=O)[O-]`, each letter representing different atoms or groups of atoms. Parentheses and other non-alphabet characters represent types of bonds and structures. The strings are generated using OEChem, a proprietary software library obtained from OpenEye Scientific Software. OEChem software was chosen for this work as they their software is available python (C++, Java and Perl among others are also available). The classes used from this library were as follows,

`ifs.oemolistream()`, used to create an input and output file stream (`ifs` and `ofs` here). The input file stream is used to read an SD file.

A `for` loop is used to iterate over the SD file.

```
for mol in ifs.GetOEGraphMols():
```

As the loop iterates over the molecules, the method OECreateCanSmiString(mol) generates the SMILES string, e.g.

```
Smiles = OECreateCanSmiString(mol)
```

The SMILES data is set (in memory) in the molecule and then written to the file using the output file stream.

```
OESetSDData(mol, "SMI", Smiles)
OEWriteMolecule(ofs, mol)
```

This code can also be used to generate and output a string which can be stored separate to the file or database in the case of this work. When inserting SMILES strings, it is also possible to insert tags which contain other data simultaneously. An example of an SMILES string tag and an information tag in a SD files are given here,

```
M  END

> <$SMI>
CC(=O)NCCCCCC(=O)[O-]

>  <fm_tagname>
ZINC01683177_000000000621

$$$$
```

Section 7.04 *Preparation of Receptor files*

The eventual target receptor files for this work will be provided by the project initiator Dr A Chubb in RCSI[42]. For test and development purposed, 400 receptor files were downloaded from the CCDC Astex database[vii]. Initially, receptor files were downloaded in

individually named directories e.g. test1a07 where the last four characters represent the name of the protein. The actual receptor files are contained in each individual directory and are all named protein.mol2. Other files in the individual directories contain additional reference information about the protein molecule. This data is of no interest to this project.

A python script was used to manipulate and rename these files and involved the following steps.

1. A backup copy of the entire group of files is made.
2. Each protein file is renamed using the (significant) last four characters from the folder name and an appendage of the file extension `.mol2`.
3. Each renamed file is then copied to a separate storage directory.
4. Using the file name as a primary key, the ASCII text files were stored to a database table.

Section 7.05 Changes to eHiTS program for security of distribution

eHiTS is available to academia and 'not for profit' work using a license system. Licensing requires the program contacting a license server at the simbiosys.ca website. eHiTS can screen files in .mol2 and .sdf formats as well as many other standard and widely available molecule file formats. During a screening run, it uses its internal conversion components to create tagged molecular ASCII (.tma) and tagged molecular binary (.tmb) formats. These tma and tmb files contain the data which is actually used in the screening calculation by the program. These formats are proprietary to Simbiosys/eHiTS and the later is in binary format and is not human readable.

eHiTS uses a shell script and arguments to this to automate and control the sequence of its operation. Much of the output of the program is controlled by this script, for example location of directories, the format of output files, etc. This made it possible to change this script to manipulate both the feedback at the command line and also some of the content and format of the output files.

After some negotiation and exchange of requirements, Simbiosys kindly agreed to provide us with a version of eHiTS which we were able to distribute without a licence. This was possible as the modified version for fightmalaria@home does not have the capability to convert ASCII sdf files to tmb format. It can only process tmb format files directly and using ASCII files directly will give an error. Conversion work must still be carried out by a full version of eHiTS, for which Simbiosys will control the license. The agreement allowing this executable to be distributed without the need for it to make contact with a license server is significant. Security around all computers with access to grid user interfaces is strictly controlled[43]. This means that access to the internet would not be a reasonable aim for software running on a worker node. The license agreement with Simbiosys means that the executable would be much simpler to deploy in a grid environment.

Section 7.06 *Database Changes made to Accommodate the eHiTS changes*

Change to the eHiTS program meant that earlier work carried out on parsing large ligand files had to be revisited. Conversion of the (ASCII) SD file format is carried out by the original and licensed version of eHiTS. These file had to be stored in the MySQL database in binary format (using binary large object or MEDIUMBLOB data type). Conversion and storage in binary format of the receptor files was also necessary. This new format required a different processing sequence. A short python script was written to add two tags to the previously parsed individual ligand files. The sequence is as follows,

Ligand files with tags are stored in a local directory.
The original version of eHiTS is used to convert individual ligand files.
Converted files are then concatenate to make multiple ligands in binary (tmb) format.

At this point, it is envisaged that the number of ligands in each file will be 100.
This input has to be given at the point of creation of this block tmb file.

Section 7.07 *Using Tags with tmb binary files*

The tmb binary file format is not human readable. However, in order to carry the tagged data needed to form the best_scores.txt file, it must contain the *fm_nametag* inserted when the files were prepared. Any tags which are needed for screening must be inserted while the file is in the ASCII state. Simbiosys have made modifications to the eHiTS program to allow it to read and use the data inside tags within the binary files.

Section 7.08 Database schema

The storage of data and creation of work units require use both numerical, text and binary data. Data types CHAR, INTEGER, TEXT and MEDIUMTEXT were used where appropriate for identification numbers and storage of file names, etc. For binary storage there are four different file formats available. These are BINARY, TINYBLOB, MEDIUMBLOB and LONGBLOB. BINARY and TINYBLOB are both limited to 255 bytes and are not suitable. MEDIUMBLOB was most suitable for the files storage required here.

Tables required for this project are outlined as follows...

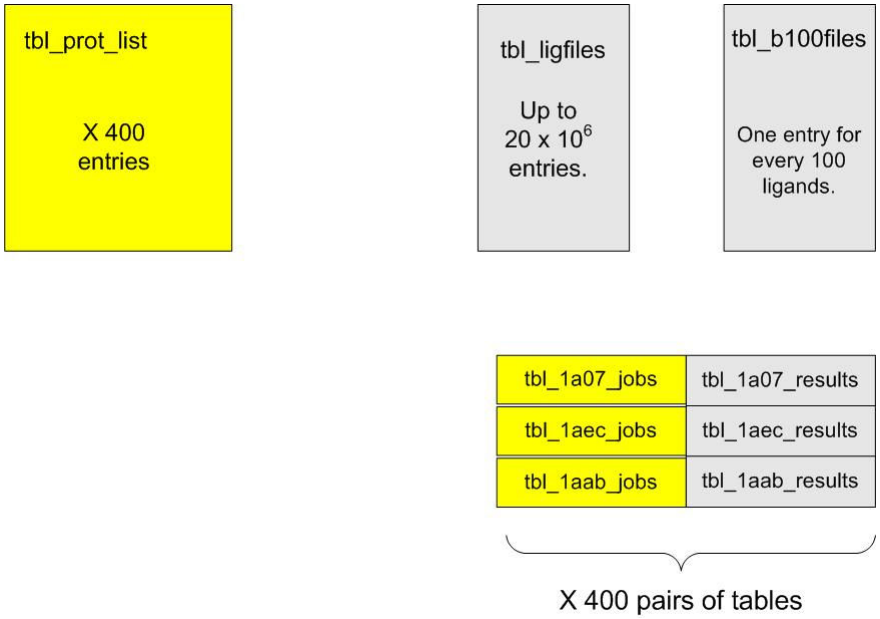


Figure 11 - Database tables

(a) Details of tbl_prot_list table

This table contains details of each individual target protein or receptor. Each entry has an id (integer) and a prot_name field by which the entry is identified. This is also the primary key. The `tmbfile` entry contains the binary representation of the receptor as generated by the convert facility in the original version of eHiTS. There is only one table of this type and it is expected that it will roughly 400 entries.

Field	Type	Null	Key	Default
id	int(4)	NO	PRI	
prot_name	char(20)	YES		NULL
comment	char(20)	YES		NULL
status	tinyint(1)	YES		0
file	mediumtext	YES		NULL
tmbfile	mediumblob	YES		NULL

Table 4 - Receptor database table

(b) Details of tbl_ligfiles table

This table contains details of all the ligands which are to be screened against the proteins listed in the `tbl_prot_list` table. It contains a unique id entry (primary key) and ligands in both ASCII and binary file formats are stored here. Although the file data stored here is not used for generation of work units for distribution screening, they are used in the preparation of data for making such work units. This table also contains a field called `smi_string`. This is a string does not play any part in the generation of work units. It is intended as a backup of the data in case of any traceability problems which may arise. An identical SMILES string is inserted in to the SD ASCII file and is found in the smi_str tag.

Field	Type	Null	Key	Default
id	bigint(20)	NO	PRI	
sdf_file	text	YES		NULL
b100	char(20)	YES		NULL
comment	char(15)	YES		NULL
smi_str	varchar(140)	YES		NULL

Table 5 - Ligand database table

(c) Details of tbl_b100_files table

This table contains similar data to the tbl_ligfiles above but instead contains block of 100 individual ligands. Each entry in this table represents a work unit when coupled with a protein file. When generating a work unit, data is read directly from the tmbfile field. Data for this table is generated by reading individual file data from tbl_ligfiles. It is then converted to binary format using the original version of eHiTS. This output is then stored in the b_file column. Using a block size of 100 ligands, this table will have one entry for every 100 files in the tbl_ligfiles table. When the Master Process (MP) is generating a work unit, the id field in the table tbl_<prot_name>_jobs corresponds to the `id` field in this table.

Field	Type	Null	Key	Default
id	bigint(20)	NO	PRI	
comment	char(50)	YES		NULL
filename	char(20)	YES		NULL
b_file	mediumtext	YES		NULL
tmbfile	mediumblob	YES		NULL

Table 6 - Database table of ligand blocks

(d) Details of tbl_<*prot_name*>_jobs table

Each protein listed in the tbl_prot_list table has an equivalent table which contains details of all the work units for that protein. These tables are called tbl_<prot_name>_jobs (where prot_name is the data in the prot_name field of a protein entry). An example of such a table is tbl_1a07_jobs. Each entry in this jobs table contains a reference to an entry in the table tbl_b100files. The b_file entry in this table corresponds to the id field in table tbl_b100files. The tbl_<prot_name>_jobs table is described as follows.

Field	Type	Null	Key	Default
id	int(6)	NO	PRI	
b_file	int(6)	YES		NULL
start_1	tinyint(1)	YES		NULL
start_2	tinyint(1)	YES		NULL
comp_1	tinyint(1)	YES		NULL
comp_2	tinyint(1)	YES		NULL

Table 7 - Database table of work-units for each protein

For redundancy, each work unit is generated twice. Fields start_1 and _2 are updated as each work unit is generated. The comp_1 and _2 fields default to zero. The comp_n fields are updated when the appropriate results file for the job is returned and processed correctly. In the future this can be used to generate a list of jobs to be generated for rework, i.e. the start_n field can be reset to 0 where comp_n has not been updated after a specific period of time.

Fields start_1 and _2 are used when determining the name of a work unit. If start_1 is 0, then the work unit will be appended with the letter 'a', if start_1 is already set to 1, then the work unit will be appended with the letter 'b'. Fields comp_1 and _2 correspond to start_1 and _2.

There are roughly 400 such tables and each will have the same number of entries in the table `tbl_b100files`.

(e) Details of tbl_<*prot_name*>_results table

Each receptor also has a table for its results called `tbl_<prot_name>_results`. These tables are described as follows...

Field	Type	Null	Default
ligand	bigint(20)	YES	NULL
score_1	decimal(6,4)	YES	NULL
score_2	decimal(6,4)	YES	NULL
score_aux	decimal(6,4)	YES	NULL

Table 8 - Database table for protein results

The name of the protein is only indicated by the name of the table, and this name is used in all SQL statements to update this table. It is envisaged that each such table will contain an entry for each individual ligand in the table *`tbl_ligfiles`*. A sample of some results in this file is given here;

ligand	score_1	score_2	score_aux	
1	-3.707	-3.707	NULL	
2	-1.1	-1.1	NULL	
6	-3.349	-3.349	NULL	
5	-2.715	-2.715	NULL	
8	-3.631	-3.631	NULL	
7	-2.582	-2.582	NULL	
10	-4.549	-4.549	NULL	

Table 9 - Sample content from table 8

Data for updates to this table are parsed from a results file as passed back from the BOINC server. There is also a `score_aux` field which can be used for re-work at a later stage.

Section 7.09 Rationale for Master Process

The design aim was to keep the core parts of the process as simple as possible while allowing separate parts of it to be extensible, reusable and scalable at a later date. Fig 12 illustrates where the Master Process and the database lie in the over all system.

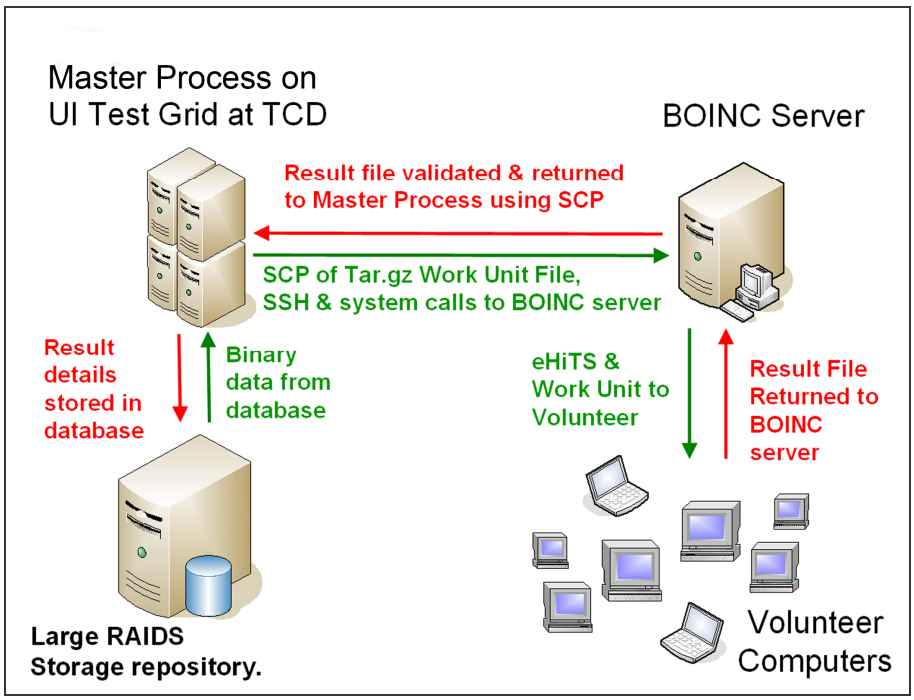


Figure 12 - Technical architecture of project

(a) Design Considerations for the Master Process

Fault tolerance: All IO read and write transactions, remote connections, etc. are nested in `try: except:` braces to allow for network or storage failures.

Distributed and Massively parallel: The Master Process is designed so that it can be replicated and run on more than one computer. This will enable many instances of it to run and hence speed up the dissemination of work units.

Well organized: The code and classes are designed so that as far as possible they are modular. This will allow changes and extensions for future work.

Self-repairing and contingent on failures: Although not a primary consideration for a working prototype, the result handling is codes in such a way to archive for trouble shooting. Provision is also left for logging of errors at critical points in the code.

Designed out of simple components: Taking a leaf from Globus 4 Toolkit, readily available components and libraries are used. E.g. freely available and unmodified python libraries were used only. PyXML, fpconst, SOAPpy were also installed for future work in the area of web services.

The Master Process (MP) is within the grid infrastructure. It has two functions.

- It generates work units for dissemination to the remote BOINC server. It used SCP and SSH calls to move and activate the work units on the remote server. This process is called ProtoPlanner.
- It constantly monitors a directory for the presence of result files returned from the BOINC server. On finding these, it validates their contents and stores results therein to the database. This process is called ProtoResults.

(b) ProtoPlanner Daemon

JobsDaemon.py is started by issuing the command `python JobsDaemon.py <interval>` at the command line in the `fm_py_src/ProtoPlanner` directory. The interval argument is in seconds and means that the process of selecting and creating a work unit tar file, and subsequently transferring it to the BOINC server will take place on this interval. The minimum time that can currently be set is 30 seconds. If no parameter is passed, this defaults to 30 seconds.

JobsDaemon.py contains a `GetJobs()` class which takes a thread as an argument. This class contains a `run`() method. When called, the `GetNextJobs.main()` is called and the thread is immediately put to sleep for the time interval passed in.

GetNextJobs.py: Calling `GetNextJobs()` initiates the creation of a work unit. This is procedural code which uses the database to determine the make up and name of the job to be generated. It uses a `Boolean` value throughout the sequence to determine whether the sequence of events proceeds or not.

The steps are as follows,

- The id number and name of the first protein for which screen is not complete is selected.
- Using the prot_name returned from the above result, a block file (b_file) and both status flags are returned from the table tbl_<prot_name>_jobs table for that particular protein.
- Based on what is returned from the above queries, the name of the tar file is determined. The tar file name is determined the name of the protein, the block file number (i.e. the job number) and whether the job is first or second work unit for that block file.
- Using the tarfile and gzip libraries, a work unit tarfile is created. The protein and block file are compressed and added to this. This file is stored in a temporary directory.
- Using SCP with public and private keys in place, the work-unit is copied to the BOINC server.
- Invoke methods on the BOINC server to move the file and also to activate the file to become a work unit within the BOINC server.
- Updating job as start_1 or _2 = 1 (True) in database if the work unit creation is successful.
- Close database objects, remove all remaining files stored on the MP computer.

(c) ProtoResults Daemon

The purpose of the ProtoResutls program is to process result files returned for the BOINC after screening. The file structure to support the collection, processing and archiving of these files contains three directories, `results_store, results_quarantine` and `results_complete`. These are located in `//opt/fightmalaria/` directory at the top of the file system on UI testgrid computer.

Result files are staged on the machine in the `result_store/`. The `opt/fightmalaria/` directory has group ownership set to allow accessible by any user who has access to the machine. This was necessary to allow the BOINC server to copy files here.

ResultsDaemon.py is started by issuing `python ResultsDaemon.py <interval>` at the command line in the `fm_py_src/ProtoResults` directory. This program checks periodically for the presence of newly arrived result files on this interval by calling CheckForResults.main().

CheckForResults.main() checks for the presence of files in the `//opt/fightmalaria/result_store/` folder by making a list of files contained there. If no results are present, it simply exits and prints a short message to the screen stating this.

If a file is found, it is opened checked for patterns to ensure that it is a valid result file. The following is an example of the head of a valid result file,

```
Receptor: 1aec
-1.374 Name: ZINC03775300_000000000844;
-1.272 Name: ZINC01683177_000000000621;
```

The code reads the first nine characters of line one. It checks that these characters 0-9 are *equal* to '`Receptor:`'. If this check returns *True*, characters 0-6 of the next line are verified as being of numeric float type. If both these conditions are met, a `task` object is

instantiated with an attribute of the file object being processed. If either of these conditions is not met, processing of the file is aborted and the file is moved to the `result_quarantine` directory.

If both validation checks return *True*, the task object is passed to a method called ParseResFile() in the ResultFileProcess class. The receptor name on line one is extracted, (`1aec` in the example shown here). An object of the class ProtLine is instantiated. This is passed to a method in this class called verify_receptor() for verification that it is a valid receptor and that it is present in the database. Following this, the subsequent lines are processed using objects of the LigLine class. An instance of the LigLine class has attributes which contain the receptor name, the ligand name and the score.

Before a line containing a score and ligand can be processed, it must be determined whether it is a first or subsequent result returned for that ligand and receptor combination. The data contained in the line is passed to a method `determine_which_score()`. This returns a string value of `score_1, _2` or `score_aux`. This decision is based on the presence of a previously recorded score for this ligand against the receptor in question. This content of this string determines which method is called to update the results table for that receptor in the database.

(d) Updating of comp_1 and comp_2 in table tbl_<prot_name>_jobs

Files returned here are named in accordance with the name of the work unit for which they represent results. Each file containing the receptor name, ligand names and scores. The constituents of the file name are also parsed and details used to update the `comp_1` and `_2` fields in the database table `tbl_<prot_name>_jobs`.

(e) Interaction with the Database

A class called `DBaseUtils()` is used for all interactions between the MP and the database. This class uses the a Python wrapper called `MySQLdb` for creating all objects

required for the database interface. `DBaseUtils.py` contains the following methods.

create_conn() and *create_cursor()* creates a connection and cursor object. When a SQL statement is executed, the wrapper must be passed both these object. Only one instance of a connection is established but several instances of cursor may be instantiated if needed.

close_cursor() and *close_conn()* terminate the connection with the database server and destroy the cursor object respectively.

execute_sql_void() is used here to execute a SQL statement where no return value is required, for example an update query.

execute_sql_ret_one() is used where a result set is returned to the MP. Python uses a data structure called a tuple to return a result set. This is an immutable object which contains commas, parentheses and formatted data. The MySQLdb methods `fetchone`() and `fetchall`() are used to parse individual rows from the result set.

Section 7.10 *Configuration file and access to it*

A configuration file is used to simplify movement of the code from computer to computer. This is, `fm_parameters.cdg` is stored in `fightmalaria/m_py_src/config/` directory. Although using one directory for the entire installation has minimised the reliance on this file, there are some variables which may change outside of the scope of the installation directory. The result directories used in `/opt/fightmalaria/` are an example of this.

This file contains paths to folders used by the ProtoPlanner, ProtoResults programs, names of servers where files are sent e.g. BOINC. It also contains details for connection to the database. Access to this file is by the `ConfigParser` library in python.

The fm_parameters.cfg file contains four groups of variables;

- [planner_folder_paths] Paths to temporary directories used during work unit generation.
- [result_folder_path] A staging or storage area for results being returned.
- [database] All log in details for access to the database on cagraidsvr03. The password to the database is not stored here. The password file is stored in the same folder but is hidden and read only by the user (permissions 400).
- [molecule_format] Details of variables which change when the two different versions of eHiTS are used on the BOINC clients.
-
- The path to this configuration file is required in the following python files,
- GetNextJob.py in the ProtoPlanner directory.
- CheckForResults in the ProtoResults directory.

This path must be set each time the program is moved or installed on a different machine as access to the `fm_parameters.cfg` file is not possible until this is done. In both files, this change is at the following code near the start of the file...

```
config.read
('/home/lavinp/fightmalaria/fm_py_src/config/fm_parameters.cfg')
```

Article VIII. BOINC Implementation

The application code, the daemons, and the work generators for this project are developed with the BOINC software platform API in the C/C++ programming language. The work-unit and result templates are coded in XML in compliance with the BOINC specifications. A number of changes to the suggested implementation of a project, as discussed in the section 6, Overview of BOINC, are implemented in this project to accommodate some individual features that it possesses. The details of these changes are discussed in the following sub-sections where relevant.

The ongoing liaising with Symbiosys Inc. regarding the release of a version of eHiTS that would be specific to this project is discussed in section 7.05. Over the course of the development of this project, the greater part of the time was spent developing with a generic version of eHiTS, and it was not until near the conclusion of the project development that the tailored version of eHiTS could be integrated into the implementation. Therefore, the logic that was applied to integrate the two versions of eHiTS into the project is discussed.

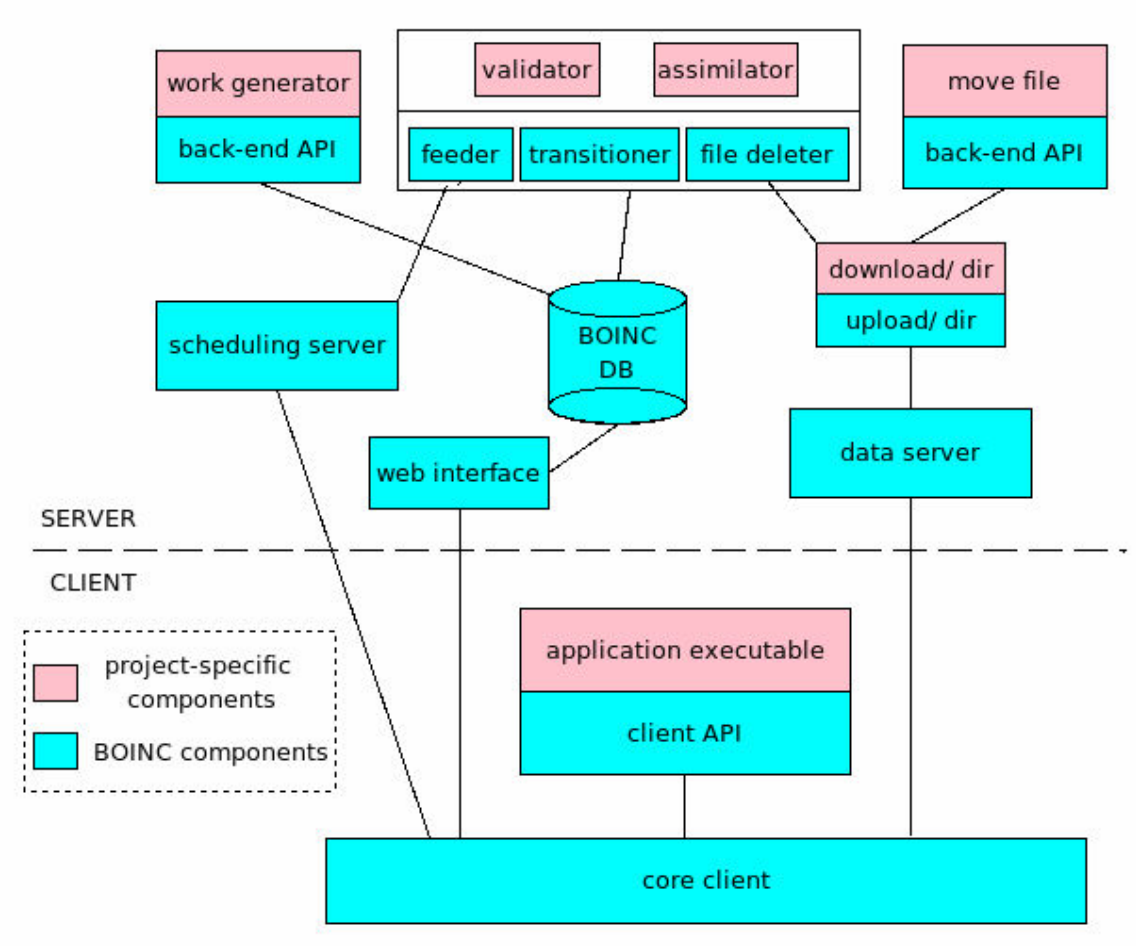


Figure 13 - FightMalaria@home architecture

Section 8.01 Application platform

Information regarding application platforms can be found in section 6.05 and section 6.05(c). It was decided to only serve to BOINC clients that were operating on a Linux, i686-pc-linux-gnu, environment for the reason that eHiTS, the biological screening software, is currently only available to run on Linux. As only versions of the application are released under the Linux platform, only clients that are run strictly on Linux are permitted to connect to the project. The possible expansion of the project to accommodate other platforms is discussed in section 14.10, under the area of Future Work.

Section 8.02 Application version

The details of releasing an application version are discussed in detail in section 6.05(b). Firstly, for the purpose of the generic version of eHiTS, three addition files are required for download along with the application executable. The four files are included in a directory under the name of the application executable. They are the application executable, the eHiTS binary file itself, the eHiTS license file, and a text file containing changes to the eHiTS SHELL script. The changes to the eHiTS script are discussed in section 7.05.

For the purpose of the tailored eHiTS version, only the eHiTS binary file needs to be included in the download/ directory with the application executable. This is because the new eHiTS version requires no licensing, and the changes that were in the text file, that is described in the previous paragraph, are supplied with in the new eHiTS version.

Section 8.03 Application code

The following is a step by step description of the mechanics of the application code. As described in the previous section, the application had to be developed with two versions of eHiTS, therefore aspects of the two contingencies are described at various points in the following sections.

(a) Command line arguments

As the application is in the form of an executable, there is a main(int argc, char **argv) function. Five command line arguments are passed upon execution, and they are as follows:

- Relative path
- Ligand file
- Receptor file
- Tag name
- Tarball name (contains the ligand and receptor files)

The relative path is necessary to enable the environment, in which the application is running, to be changed to the project directory (see section 6.11(a)) by using the chdir(char*) command. This path is not hard-coded into the code to anticipate any changes in the structure of the BOINC client. Both the ligand and receptor filenames are specified so that they can be called in the eHiTS command by way of a system call. System calls are performed using the system(char*) function. The purpose of the tag name is explained in section 5.11(c). It is called along with the ligand and receptor filenames in the eHiTS command. Due to the logical naming system (see section 6.07), the tarball name is the same for each work-unit despite the physical name being different. The logical name is passed in as an argument to allow greater flexibility for changing the logical naming in the future.

(b) BOINC initialization and logging

Initially, the boinc_init() and boinc_dignostics(int) functions (see section 6.05(a) for details) are called in the application code. A single output file is uploaded to the BOINC server on completion of each work-unit, so a call is made to the two BOINC API functions: boinc_resolve_filename(char*, char*, int), and boinc_fopen(char*, char*). A char pointer representing the logical filename is inputted to boinc_resolve_filename(char*, char*, int), and a char pointer is returned containing the path to the physical file. The logical filename, best_scores.txt, is used, as it is the filename that is specified in the result template file (see section 6.07(b)). The contents of this file are returned when the work-unit is finished. The boinc_fopen(char*, char*) function enables the file to be opened.

A log file is then opened. This file is written to each time a command is called during execution. Should there be a problem returning a valid results file upon completion of the execution, the contents of this log file are written to the file that is to be uploaded. This means that the probable reason for work-units failing can be ascertained on the server-side. A textual code is inserted into the first line of the file so that the work-unit can easily be recognized to have failed. This code is of the form: “ERROR!”

(c) Directory change

By way of the chdir(char*) command and the relative path that is passed as a command line argument, the environment is changed from the slots directory to the projects directory of the BOINC client. This is necessary even though the BOINC client attempts to give access to all the relevant files from the slots directory. However, it does this on the assumption that the downloaded files are accessed solely for reading, and by way of the boinc_resolve_filename(…) function. BOINC relies upon its own soft-link XML format to relate files in the slots directory to the actual files in the projects directory. As files need to be accessed directly with system calls, the environment has to be the projects directory.

(d) Extracting eHiTS

If a work-unit is received for an application that has already run on the client, then all of the relevant files would already present in projects directory (see section 6.11(a)). Not only this, but if they are altered or expanded, then their state is also maintained. The eHiTS executable is downloaded in a binary file format. Calling this binary file creates an eHiTS/ directory and associated soft-links to its respective scripts. This extraction has to be carried out only when an application is first downloaded. To ensure that eHiTS is not unnecessarily extracted each time that a work-unit is processed, a search is done in the code of the contents of the projects directory to see if it contains the ehits.sh SHELL script. If it is not present, then a system call is performed to extract eHiTS.

(e) Licensing and eHiTS modification

These steps are only relevant to the generic version of eHiTS. They are carried out under the same circumstances as the eHiTS extraction in the previous section.

The generic version of eHiTS requires a license file so that the software cannot be used illegally. The license file needs to be in the eHiTS/ directory to be recognized. A system call is made to copy the license file to the directory.

As the amount of data that is produced in the eHiTS result file is too verbose, measures are taken to reduce this data to an amount sufficient for this project (see section 7.05). The text file that is downloaded with the application contains the changes that need to be made to the eHiTS script. The contents of the file replace a section of code in the ehits.sh script. To achieve this, the ehits.sh script is parsed by the application code. The contents of the script are written to a temporary file, except for the section that needs to be replaced. The contents of the changes file are written to the temporary file while the section that is to be replaced is being parsed. When the temporary file has been completed, its contents are re-written to the ehits.sh script.

(f) Extracting the ligand and receptor files

A system call is made to extract the ligand and the receptor from the tarball that is downloaded with each work-unit. It is with these biological text files that the eHiTS software can be invoked.

(g) Running eHiTS

The eHiTS software is also called using a system call. The syntax of the call for the generic version of eHiTS is of the form:

```
ehits.sh -ligand [*.sdf] -receptor [*.mol2] –tagname
“tagname” -workdir [workdir]
```

The .sdf ligand file may contain information relating to one or more ligands. It takes roughly 20 minutes to process each ligand in the .sdf file.

The system call for the tailored version of eHiTS is of the form:

```
ehits.sh -ligand [*.tmb] -receptor [*.tmb] -tagname “tagname”
-workdir [workdir]
```

Similar to the .sdf file that is mentioned above, the receptor .tmb file can contain information on many ligands. The tailored version of eHiTS runs marginally faster than the generic version, due to refinements in the amount of processing.

The -workdir argument is set to the current directory, './'. This means that eHiTS puts the processing results into the directory from which the eHiTS command is called. Therefore, all of the results are placed in the projects directory. They are placed here because they are automatically deleted from the BOINC client in the event of a user detaching from the project. In this instance, unwanted files are not left over on the client machine. This measure is also taken in anticipation of later versions of the BOINC client possibly running

in a sandbox environment. In this event, having eHiTS results files in the projects directory rules out the risk of interference from the sandboxing.

(h) The eHiTS results file

The path to the eHiTS file that contains the top scores, called best_scores.txt, for each ligand has to be resolved. It is in the following format:

```
BOINC/projects/[host_project]/[receptor]/[ligand]/best_scores
.txt
```

This path is constructed using the command line arguments for the receptor and the ligand. The file is parsed to ascertain whether the file is valid and can ultimately be sent back to the BOINC server. If the file is valid, then it can be read and written to the file under the logical name of best_scores.txt that is opened using the boinc_fopen(char*, char*) API call. Should the results file be invalid, the contents of the log file are read and written to the logically-named best_scores.txt file.

Section 8.04 Work-unit template

A description of the XML schema for the design of a work-unit template is given in section 6.07(a). The work-unit template that is used in this project is of the form:

```
<file_info>
        <number>0</number>
        <report_on_rpc/>
        <max_nbytes>9000<max_nbytes/>
</file_info>

<workunit>
```

```
        <file_ref>
                <file_number>0</file_number>
                <open_name>zipped.tar.gz</open_name>
                <copy_file/>
        </file_ref>

        <min_quorum> 1 </min_quorum>
        <target_nresults> 1 </target_nresults>
        <max_error_results> 1 </max_error_results>
        <max_success_results> 1 </max_success_results>
        <max_total_results> 1 </max_total_results>
</workunit>
```

Only one generic template is needed, and it is associated with each work-unit that is created. One input tarball file is associated with each work-unit, and it can be seen to be described above using the <file_info> and <file_ref> tags. The file is given the generic name, zipped.tar.gz. It is under this name that the tarball is downloaded to the BOINC client.

Redundancy is implemented in another area of the project. It is for this reason that no redundancy is implemented in BOINC. The <min_quorum> and <target_nresults> values are set to one. This means that each work-unit is only sent out to a BOINC client once, and that a canonical result is found irrespective of the data returned with the result. Error handling is conducted in other area of the implementation of this project.

Section 8.05 Result templates

A description of the XML schema for the design of a result template is given in section 6.07(b). The result template that is used in this project is of the form:

```
<file_info>
```

```
    <name><OUTFILE_0/></name>
    <generated_locally/>
    <upload_when_present/>
    <url>

     http://binabol.cs.tcd.ie:8085/proto_cgi/file_upload_hand
ler
    </url>
    <max_nbytes>200000</max_nbytes>
    <sticky/>
</file_info>

<result>
    <file_ref>
        <file_name><OUTFILE_0/></file_name>
        <open_name>best_scores.txt</open_name>
    </file_ref>
</result>
```

The <open_name> tag contains the text, best_scores.txt. This is the logical name for the file that is opened using the BOINC API functions that are mentioned in section 6.05(a). This file contains either the correct scores for the biological matching, or the log data for the application, depending on the outcome of the execution. The details of creating and naming the file that this template describes are given in section 6.07(b).

The upload URL is specified as being the location of the file_upload_handler CGI script that is supplied with the BOINC software.

Section 8.06 Work generator

It is suggested in the BOINC documentation that a work generator be developed as a daemon (see sections 6.10 and 6.10(e)). However, the decision for creating new work-units on the basis of the completion rate of outstanding work-units is made in a separate area of the project. Therefore, work-unit generation takes the form of system calls to an executable that is coded in C/C++. A BOINC API function is provided to create work, and it is described in section 6.06(b).

(a) Command line arguments

The main(int argc, char **argv) function of the code is passed-in several command line arguments that are used in both the creation of, and the relaying of command line arguments to the application executable of, the work-unit. Five command line arguments are passed to the code, and they are as follows:

- Ligand filename
- Receptor filename
- Tagname
- Tarball filename
- Work-unit name

(b) Environment switch

As the executable is invoked remotely, relative paths would not usually be used in calls to the BOINC API function to create work (see section 6.06(b)) to indicate the location of the result template XML file – an absolute path can be used to locate the work-unit XML file, because its contents have to be allocated to a char pointer elsewhere in the code. However, the BOINC function does not seem to accept absolute paths, despite indications in the

BOINC documentation that absolute paths are valid. Paths to the result template file, that are relative to the project root directory (as per examples in the BOINC documentation), need to be specified. To circumvent this issue, the chdir(char*) C++ function is used to switch from the environment that the executable is running in, to that of the project root directory. In this manner, relative paths to the result template file are valid, irrespective from where the executable is being called.

(c) Work-unit definition

Two instances of BOINC-defined classes, DB_APP and DB_WORKUNIT, need to be created. A function of DB_APP, called lookup(char*), needs to be called to yield the ID of the project that is being run. This ID is used to define a data-type in the DB_WORKUNIT class to associate the work-unit with a particular project. Many other data-types in the DB_WORKUNIT class can be defined to control the behaviour of the work-unit. The most important of these are the name and the redundancy settings. The name of the work-unit is received as one of the command line arguments that are mentioned section (a) above. The values that determine the redundancy of the work-unit are the min_quorum and target_nresults integers. The redundancy can also be defined explicitly in the work-unit XML template. The work-unit template and redundancy are explained in section 6.07(a).

(d) API call to create work

A char pointer array of command line arguments is constructed. These arguments are to be passed to the application executable as it is executed on the BOINC client. The pointer contains three of the arguments that are mentioned above in section (a):

- Ligand filename
- Receptor filename
- Tagname

It also contains two additional arguments:

- Logical name for the tarball ('zipped.tar.gz')
- Relative path to the projects directory (see section 6.11(a))

The name of the tarball file that is to be downloaded with the work-unit is placed in an array of char pointers.

An instantiation of the BOINC-defined SCHED_CONFIG class is created. The contents of the config.xml file (see section 6.03(b)) are read into this class by calling its function, parse_file().

Finally, the call is made to create the work-unit by passing the following arguments:

- DB_WORKUNIT instantiation
- char pointer to work-unit template
- relative path to result template
- char pointer array of input files
- number of input files in the char pointer array previous
- SCHED_CONFIG instantiation
- char pointer of command line arguments

Section 8.07 Move files

An executable that is coded in C/C++ is called to move a file from a BOINC-independent temp/ directory to the download/ directory, under its appropriate sub-directory in the download/ directory hierarchy. This executable is created to take advantage of the BOINC API function for resolving file paths in the download/ directory, and to reduce the complexity of having to move the files remotely with the BOINC binary command. The API and binary tools are explained in sections 6.03(c) and 6.06(a) respectively.

The executable takes a single command line argument. It is the name of the file in the temp/

directory that is to be moved to the download/ directory. As well as an empty char pointer and other arguments, the filename is passed into the API function. The char pointer is resolved to be the absolute file path that the new file is going to occupy in the download/ directory hierarchy.

Finally, a system call is executed to copy the file from the temp/ directory to its destination path in the download/ directory as indicated by the resolved absolute file path.

Section 8.08 Daemons

Daemons relating to a BOINC project are discussed in section 6.10. The daemons that are implemented in this project are:

- Assimilator
- Validator

The reasons for not implementing a work generator as a daemon are described in section 8.06.

(a) Validator

The expectation of the Validator daemon, as defined by BOINC, is to analyse incoming redundant (see section 6.07(a)) results, and to select a canonical result from them. However, as described in section 8.04, the option of redundancy in BOINC is not utilised. Therefore, the purpose of the Validator becomes trivial. Its sole purpose it to provide empty functions, the calling of which enable the Transitioner to prepare the results for processing by the Assimilator.

(b) Assimilator

The Assimilator daemon is developed in conjunction with BOINC-specific code. The BOINC-independent code is initiated when a call is made to the function:

```
assimilate_handler(WORKUNIT& wu, vector<RESULT>& results, RESULT& canonical_result)
```

As there is no redundancy used for work-units (see section 8.04), the canonical_result, of type RESULT, that is passed into the function always contains a canonical result. The fact whether the content in the results file, that relates to that RESULT instance, is valid or not is determined in other areas of the project.

Back-end API functions are provide with BOINC (see section 6.03(c)) to resolve the path to the results file in the hierarchy of the upload/ directory. The file in the upload/ directory is read and re-written to a BOINC-independent output/ directory. This is due to the fact that the File Deleter daemon (see section 6.10(c)) deletes files in the upload/ directory once their respective result is processed by the Assimilator.

Should results files need to be retrieved from the BOINC server due to unforeseen difficulties, then they would be still available and not deleted. The naming of a results file is discussed in section 6.07(b). The file that is created in the output/ directory is named solely after the work-unit, and the additional redundancy-related extensions are discarded.

Finally, the results file in the upload/ directory is copied to the remote location of the Master Process (see section 7.09) using SCP. Configuring the system so that SCP is possible is discussed in the next section

Section 8.09 SCP configuration

Asymmetric keys are generated using the ssh-keygen command. The public key is placed in the .ssh folder of the user that owns the execution rights of the Master Process. The private

key is placed in the .ssh folder of the ‘boinc’ user that operates the BOINC project. In this manner, results files can be copied securely – using SCP – to the remote Master Process machine by the Assimilator daemon automatically, or in other words, without having to specify a password.

Article IX. Problems Encountered (I)

The follow section explains the various problems that were encountered during the progress of this project and in some case what was done to overcome them.

(a) File Handling with Large Number of Files

Preparation of multiple ligand from relatively small individual files involves handling large amounts of these small files. Roughly 90 files containing over 2,020,000 molecules (7.9GB) were downloaded and parsed. When parsing these to individual files, the OS became increasingly slow when the number of files in any single folder was over 30,000. For this reason, each of the 90 files downloaded were parsed in to individual directories. This kept the number of file in each directory to about 25,000 which gave acceptable performance with no debilitating delays when reading and writing.

(b) Maximum Packet Size for Transactions with Database

The default maximum packet size transmissible from either a MySQL client or server instance is 11048567 bytes. This limit came to light when storing receptor files larger than this size to the database as files were stored but were truncated at approximately 1MB long. This variable can be changed by adding or altering the following line in the file my.cnf to read...

```
set-variable = max_allowed_packet =16M
```

This must be done on the client and the server instances.

(c) Loading binaries to the database

Molecules in tmb format, once created have to be stored in a database table for 'on the fly' retrieval during work unit creation. The MEDIUMBLOB can store file of length up to 2 x 1024 bytes. Storing binary strings using SQL statements passed using python and the MySQLdb module requires that all characters must be escaped. This became apparent when SQL errors for *malformed SQL strings* occurred. This problem is overcome by using the `_mysql.escape_string` module. On inspection, it can be seen that this module adds characters to escape characters which would otherwise interfere with how the MySQL module in python handles an SQL string.

This problem occurs when storing binary files. An example of what causes this is the ASCII new line character \n is represented by the bit sequence 00001010. In a binary file (say a 500KB receptor file) it is possible that at some point, this sequence of bits will appear at some multiple of 8bit (1byte) offset from the start of the file. This may occur randomly and it is purely coincidental and bears no relation to the new line character. This and other bit patterns which show up as other ASCII control characters in SQL statements and cause errors in the MySQLdb module and in the MySQL interface.

(d) Slash Characters in Protein Files

A small number of the protein files from the CCDC Astex contain forward slash (/) characters. When inserting the ASCII versions of the protein files to the database, these characters caused SQL errors. To overcome this, all the files were subjected to a two stage character escaping routine. On retrieval, the files were also put through a reverse form of this. This enabled all the files to be stored without errors. The manually written code was as follows,

```
file_0 = file_obj.read()
file_1 = file_0.replace("\\","\\\\")
file_2 = file_1.replace("'","\\'")
```

```
# write to database
# retrieve from database

resultbuffer_0 = result1_1[0]
resultbuffer_1 = resultbuffer_0.replace("\\'","'")
resultbuffer_2 = resultbuffer_1.replace("\\\\","\\")

f.write(resultbuffer_2)
```

A short python script was written to retrieve these files and create a temporary version of these on the local hard drive. Each file was then subjected to the shell command cmp to verify that the stored file was identical.

(e) Linux

A comprehensive familiarity with the Linux operating system was lacking for the early development of this project. Time was spent solving system administration issues which would have been more quickly resolved if more experience have previously been acquired. Several issues arose when installing software and configuring the Linux environment. An appreciation of the file directory structure and environment variables of the Linux operating system is essential knowledge for installing and removing software.

(f) Compatibility of Python versions and libraries

Early development work in this project was done using Python 2.2.3. This version was chosen for compatibility with the current version deployed on UI.testgrid. Compatibility issues became a problem when

Article X. Problems Encountered (II)

The follow section outlines the various problems that were encountered in the course of this project. In summary it is fair to say that some problems were due to a lack of experience in the environment of the particular operating system and with the tools that were used; others were due to some inherent weaknesses in those tools.

Section 10.01 Linux

A comprehensive familiarity with the Linux operating system was essential for the development of this project. Time was spent solving problems that would have been more quickly resolved if more experience have previously been acquired. Several issues arose when installing software and configuring the Linux environment.

Section 10.02 Issues with BOINC

A lot of time was spent coming to terms with the implementation of a BOINC project. Needless hassle is placed upon project developers by the administrators of BOINC. Firstly, the documentation does not follow the work-flow involved in setting-up a BOINC project. It is not made clear which aspects of a project have to be implemented in tandem with the BOINC infrastructure, or which aspects already exist in the BOINC infrastructure. Even when it is apparent that BOINC-independent work has to be done, a lot of implied knowledge is inferred in the documentation. The documentation is not written from a perspective of, or directed at, someone who has no prior knowledge of the workings of a BOINC project. The consequence of this is that a trial and error approach has to be adopted, when more elaborate documentation would have encouraged the adoption of a more logical and productive approach.

(a) Documentation

A good example of where these documentation issues can be seen is when coding a work generator. This is only possible by defining the variables, and invoking the functions of, BOINC-developed C/C++ classes. Insufficient information regarding these classes is provided in the documentation, so debugging these functions involves having to read and assimilate the BOINC code, instead of being able to treat the BOINC software as a 'black-box', which is a viable option for a generic toolkit such as this.

Section 8.06 gives a description of the implementation of the work generator for this project. The function to create work is described in section 6.06(b), and has to be used in the design of a work generator. The documentation states that an absolute path can be provided to indicate the location of a certain XML template. However, the function would not accept an absolute path, and alternative measures had to be taken to circumvent these issues.

(b) Documentation - Calling proprietary software in BOINC client

It is intended that all code – typically written in C/C++ – that is invoked on the BOINC clients be coded by the developer(s) of a BOINC project, and that this code avails of the API calls to interact with the environment of the client. However, this project is constricted by having to invoke proprietary software (eHiTS) with system calls. This fact resulted in many complications with regard to achieving successful execution, and the return of valid processing results from the clients to the server. For reasons explained in section 8.03(c), an environment switch has to be carried out in the application code. Although an appropriate solution is implemented in the code, many attempts had to be made to discover the solution. This was due to the fact that the documentation does not go into detail on the dynamics, architecture, and environment that are evident on the client when the application executable is being invoked. However, as BOINC was being used in this project in a manner outside of what was originally intended by its developers, more elaborate documentation could not reasonably be expected, although it could be argued that the availability of such documentation would promote more understanding about BOINC among its developers.

One of the factors that contributed to forcing the environment switch, that is mentioned in section 8.03(c), is that the BOINC documentation refers to 'soft-links' that are created in the slots directory (see section 6.11(a)) that link to the actual files in the project directory (see section 6.11(a)) that have been downloaded from the server. However, as could be assumed, these 'soft-links' are not symbolic links that are common in the Linux environment. They are in fact link-files that contain an XML tag containing the location of the file to which the link-file points. This BOINC-style method of linking meant that some programming steps were taken on the assumption that the link-files were in fact symbolic soft links, however this proved to be incorrect, and some implementation efforts went wasted.

(c) Error-handling

The issue discussed in the previous section was exacerbated by poor error-handling. This level of error-handling is indicative of most aspects involved in invoking API functions in the development of a BOINC project, none more so than the execution of an application executable on the BOINC client. A lot of problems are inherent in invoking an executable on a remote host, coupled with the download and upload of input and output files that the executable both requires and produces. However, the errors that are outputted either in the console of the client, or in server-side log files, are not informative enough to enable a developer to debug the problems in anything other than a trial and error manner.

An example of such an occurrence is the discovery of a 'Could not find or stat script' error message in an Apache web server log file. This would indicate that CGI script for the scheduling server cannot be found, even if the process of altering of the schedulers.txt file (see section 6.03(d)) is correctly enacted. The cause can be deemed to be a bug in the BOINC code, as the resolution of the error is not explained in the documentation.

The solution can only be found either by chance, or by 'Google-ing' it. The problem arises as there is a blank space after the project URL in the <scheduler> tag in schedulers.txt. This data is automatically generated by BOINC, and the functioning of the BOINC project

hinges on this unnecessary black space being removed. Furthermore, should this blank space be removed, there is also a <link> tag that provides the URL of the project by way of a 'rel' attribute. This tag is also generated automatically by BOINC, and also has to be removed as it is not compatible with the BOINC clients. This is also not documented.

Section 10.03 eHiTS performance on BOINC

The typical processing times of eHiTS are described in table 2. When eHiTS is invoked with the BOINC client, the processing time seems to be of the order of 3 times as long. The reason for this increase in processing time is not immediately obvious. Another issue is apparent with regard to running eHiTS on the BOINC client. The execution of eHiTS times out about 50% of time. This means that invalid results are sent back to the BOINC server. The resolution of these two issues might lie with further liaising with the proprietors of eHiTS.

Section 10.04 Conclusion

As is discussed throughout the previous section, a myriad of problems were encountered while developing a BOINC project in this implementation of this project. Many problems were the result of insufficient documentation and error-handling. Some issues were encountered due to the fact that proprietary software is invoked in the BOINC client in a manner that is contrary to what the developers of BOINC intend. The proprietary screening software proved to be inconsistent when run in the BOINC client. Progress was also partially delayed due to unfamiliarity with the Linux operating system. As a consequence of these issues some of the original goals of the project did not come to fruition.

Article XI. Analysis (I)

This section attempts to give a critical analysis of the work implemented in the Master Process when undertaking a challenge of this size.

Section 11.01 Comparison to Initial Goals

An initial aims of the research was to get an accurate, viable, efficient and scalable design. For the most part although simple in its implementation, the Master Process is these things. Increased flexibility around its operation is now necessary and aspects of where this could be done are outlined in the section Future Work (I).

A second aim of the project was to develop and test architecture and mechanisms for distributing software for screening drugs on the TCD test grid. This part is not completed although design decisions taken early on make this possible with further work. The work focused on getting successful generation of work units and getting a 'round trip' of work unit to processing and back in the form of results. However, work units are generated in the form of one file coupled with an executable and a name of an output file. This is also the format that is required for submission of jobs to a grid.

Significant time was spent on work and design of the customisation of the eHiTS program during the work although the need for this was largely unforeseen at the early stages of the project.

Section 11.02 Suitability of work units

Work units are of fixed size. This decision to generate work units was taken early in the work and did not consider the variation in file size. For a challenge of this size, the difference between good optimisation and efficiency and the lack of it could make a difference of a number of years to the completion time. The effective re-use of pre-

processed data in a grid environment with substantial storage capacities and bandwidth is outlined in the Future Work (I) section.

Section 11.03 Speed of dispatching of work units

Work unit need to be dispatched at a rate faster than they can be processed. Otherwise, available computation capacity will be under utilised. However, an over zealous dispatch of work units could possibly cause a bottleneck and over burden storage on the remote resources to which they are dispatched.

Section 11.04 Comparison to past drug screens

While similar in overall approach to some of the work done on WISDOM, the size of this challenge is significant. Not enough was done at the early design stages to utilise the experiences of these projects.

Section 11.05 Direct use of a grid environment

The implementation of the Master Process was a prototype. However, it was designed with the dispatching of work units via JDL directly to grid resources in mind. It can also be easily modified to become a web services client and dispatch work units to a Social Grid Agent. Both these areas are possible paths of expansion for this challenge. The implementation is now tested and running in an environment within a grid which means that is correctly placed for future expansion using grid resources.

Section 11.06 Master Process Database Performance

No issues arose with performance around the database used for storage of biological and results data. Tests carried out on the mock results table (containing 2 million rows) showed no detectable delay when retrieving results. This was true when searching using the integer

ligand numbers or for searching for particular scores. The database instance was a dedicated to this work. This meant that it had only one user, i.e. the Master Process.

Section 11.07 eHiTS

The customisation of eHiTS is a significant contribution both in the area of summarised scores and in the area of licensing. This product is designed for desktop and small cluster operation. Although it works well in this environment, there appears to be many areas where it could be tailored and optimised to make it a very efficient tool for use in large scale parallel and highly efficient drug screens.

Article XII. Analysis (II)

The following section gives a critical analysis of the success of the implementation of this project. Firstly, a comparison is made of how the actual implementation has fulfilled the initial goals. The success rate and consistency of the test data is discussed. The applicability of the software and programming platforms (i.e. BOINC) that were used in the project are considered. The level of success and reasons behind the project architecture are provided. A critique is made about how useful the BOINC platform actually turned out to be. The security that BOINC may offer to future volunteer users is explained. Detail is given about the reasons for the changes made in the BOINC model. Comparisons and contrasts are made between the work that has been done in this project and work that has been done in related areas. Finally, the suitability of the remote interface that has been developed between the BOINC project and the Master Process is discussed.

Section 12.01 Success of initial goals

The initial goals for the project were set out in a project proposal that was submitted in December 2007. The main goals that are described in this proposal have been achieved in the implementation of this project. However, some subsidiary goals were not completed due to time constraints.

A BOINC project has successfully been developed to administer volunteer nodes, to enable those volunteer nodes to process biological data, and to return valid computational results. Unfortunately, the BOINC project has not been integrated into a Grid environment, and a Web Front-End has not been developed to monitor the status of processing jobs. An agent was also described in the proposal to accept jobs from Grid scheduling components and return results from the BOINC server; however this aspect is yet to be implemented and is part of the future work of this project.

Section 12.02 Consistency of results

Some inconsistencies have arisen due to the unreliability of the eHiTS software when invoked with the BOINC client. The eHiTS invocation tends to fail 50% of the time due to a time-out. However, this problem can not be attributed to the quality of the implementation of this project as a zero return code is returned from the system call to eHiTS. Therefore, this is an issue that has to be resolved with the proprietary eHiTS software. Otherwise, the BOINC project and respective code operates with consistent results. Work-units are created and results are handled in a similar manner each time that processing jobs are carried out.

Section 12.03 Choice of software

There were some issues with the development of a BOINC project that are discussed in section 10.02. However, after these issues resolved, the choice of using BOINC proved to be successful, as not only has a working BOINC project been developed, but the advantages that BOINC offers to its projects in terms of computing power could make the future development of this project worthwhile and successful.

The choice of biological screening software has proved to have both positive and negative aspects. The advantages of using eHiTS are that is it free for use in this project, that is runs reasonably fast, and that its source code is available. Conversely, some of the disadvantages of using eHiTS are described in section 10.02(b). From the point of view of BOINC, a different choice of software could have been of more benefit and convenience. As described in section 10.02(b), various issues arose because of the fact that eHiTS had to be invoked using a system call. A better option would have been for the client to have been able to download the screening software as a shared library that is coded in C/C++. In this manner, the screening software could have been invoked from within the application executable code, and a lot of time could have been saved. Also, by ruling out the need for making a system call, possible unforeseen errors, for which there might be little or no error

return data, could potentially be avoided.

Section 12.04 Applicability of project architecture

The architecture of the project has been developed to be integrable with a Grid environment. The work generator executable is invoked remotely. Likewise, the results that are passed back to the BOINC server from the clients are returned to a remote location. This means that the BOINC server is suited to being installed as a Grid resource. Processing jobs and results that are passed throughout a Grid infrastructure are all formatted and described in a generic way that is suited to remote communication. A wrapper could be placed around the BOINC interface to receive processing jobs and return results, and this package could be integrated into a Grid environment.

However, the current implementation was devised as being a basic version to show that this kind of remote architecture is viable. At present, as system calls to executables on the BOINC server are made, the current setup would not instantly be integrable, or ideally integrated, into a Grid scenario. A preferred configuration for integration with Grid architecture would be to enable BOINC-powered code to be directly invocable by Globus Toolkit (GT4)-specific code. In this manner, more control is given to code, and a higher verbosity level of error-handling can be yielded.

Section 12.05 Performance of BOINC

As previously discussed, BOINC provides API functions with which to interact with its platform and environment. However, having to merge these function calls into the BOINC-independent software created many problems (see section 10.02(c)). A better method might be to provide the functionality so that BOINC-independent software could be plugged-in to the BOINC environment. In this way, BOINC software can be treated as a 'black-box', instead of having to extensively debug integrated API calls.

No scope is provided with BOINC for generating work-units over RPC. An RPC API might ease the transition of integrating BOINC to a Grid environment, as the RPC functions could be invoked using Web Services.

The expected processing times for eHiTS are described in table 2. However, when running on the BOINC client, the processing time required can increase almost three-fold. This issue is discussed in more detail in section 10.03.

Section 12.06 BOINC security

As discussed in section 6.04, public key cryptography is used to sign executables. By this means, users who run BOINC clients can verify the identity of the source of executables, and related files. A certain amount of trust must be placed in BOINC by the users. The identity of a BOINC project may be ensured by the operators of BOINC. However, the trust that can be placed in the operators of each BOINC project is uncertain. Therefore, a small amount of risk, with potentially serious consequences, is involved in attaching to a BOINC project.

Section 12.07 Changes to BOINC model

Three significant changes have been made to the model of the BOINC platform. These changes have been made so that the constraints that are set out by the specification of this project can be accommodated by a functioning BOINC project. In other words, BOINC has been adapted in the development of this project to conform to certain performance goals.

Firstly, work-units are generated and files are moved into the download/ directory from a remote source. This fact contradicts the intentions of the BOINC developers, as no RPC capability is provided to do this. Secondly, the Work Generator itself is intended to be a

daemon that generates work-units, on account of outstanding work-units being completed. However, the re-generation of work-units is carried out at the Master Process, and so having a daemon to do so would not be applicable. Finally, the Validator daemon is intended to be used to reach a consensus, or canonical result, among redundant results of work-units. However, the Validator in this project performs trivial tasks due to the fact redundancy is not utilised in the BOINC portion of this project. Redundancy has been abstracted out to the Master Process.

Section 12.08 How project compares to related work

Related work is discussed in section 2. In terms of existing and beta-stage BOINC projects, the goal of FightMalaria@home differs from the related work in the field. Even MalariaControl.net takes quite an alternative approach of using BOINC to tackle the issue of malaria. In this respect, FightMalaria@home is novel among the Volunteer Computing community.

However, the intention of integrating this BOINC project into a Grid environment overlaps with other work in the field to some extent. Researchers at CERN have implemented a system in which the submission of processing jobs is alternated between a BOINC project and a Grid Resource Allocation Manager (GRAM). Furthermore, the Lattice project at Maryland University uses Condor scheduling software to organise the distribution of work to Globus-addressable Grid resources, one resource of which is a BOINC project.

The details of how the current configuration of BOINC in this project is integrated into a Grid environment are yet to be decided. There seems to be two valid options available. One of which is to maintain the existing implementation of BOINC, and to either send work to it or to Grid resources. The other option is to slightly modify the current configuration of the BOINC project so that it can be made into a Globus-addressable resource. Both of these options are comparable to the related work that is mentioned in the previous paragraph. The former option is similar to the system that has been developed at CERN, and the latter is equivalent to the implement that has been developed at Maryland University.

Section 12.09 Suitability of interface

For the most part, the remote interface between the BOINC project and the Master Process followed a logical progression. For each work-unit that is created by calling the work generator, the work-unit name, the ligand and receptor filenames, the logical zipped file name, and the eHiTS tagname need to be provided. Including the work-unit name plus the ligand and receptor filenames is obligatory. However, it was decided to include the tagname as an argument to anticipate any future changes to the tagname in the ligand files. Having to specify the tagname as an argument means that should the tagname be changed, then no code would have to be changed in the BOINC project. Likewise, the name of the zipped file is specified so that no changes to the BOINC project code would be needed should its name be changed in the input XML template.

The methods to insert input files into the download/ directory are discussed in section 6.03(c). An executable was created that uses the API function to simplify the process of inserting a file into the download/ directory remotely. It was also envisaged that the coding of any executable would side-step any issues related to locating input files using absolute paths.

Section 12.10 Conclusion

Overall, the main goal of the project was achieved successfully. This is the implementation of an eHiTS-enable BOINC project. However, time constraints dictated that integration with the Grid could not occur. Grid integration forms a large portion of the future work for this project. A rate of 50% consistency was achieved by screening with the proprietary eHiTS on the BOINC clients. It can be concluded that the resolution of this issue is beyond the reach of this project. EHiTS had to be invoked on the BOINC client using system calls. This led to some implementation difficulties. Using screening software that is more

compatible with the C/C++ BOINC environment might resolve inconsistencies in the processing results. The design of the architecture for this project is conducive for future integration with a Grid; however the reliance on system calls has to be reduced.

Using BOINC for Volunteer Computing proved to be a success, although its strong orientation towards unreliable API functions does lead to unnecessary delays in development. As long as initial trust is placed with BOINC by volunteer users, the security of an individual BOINC project is such that the source of downloaded data can be authenticated successfully. Some adaptive changes were made to the BOINC model to suit the needs of the project. FightMalaria@home is unique among other BOINC projects. However, parallels can be drawn between the intended goals of this project, and related work in the area of integrating BOINC projects with Grids. The interface between the two major components of this project proved to be well planned, with sufficient scope given so that changes in the interface data would not force changes to be made in the project code.

Article XIII. Future Work (I)

There are some areas in this work which could be optimised and further developed in the future. These are outlined in this section.

Section 13.01 Modifications to eHiTS Program

Some further debug and testing work is required with the modified eHiTS program. Initial beta versions sent did not run on Scientific Linux 3.0x (a clone of Redhat EL3) but ran successfully and gave consistent results when run on Ubuntu 6.10. A later release of eHiTS ran successfully on SL3.0 and on Ubuntu 6.10. At this point of demonstration, the latest release of eHiTS was used.

Section 13.02 Social Grid Agents

A grid structure at resource broker level is concerned with the exchange and sharing of resources by different resource owners. There are similarities between the interactions of resource brokers and interactions in actual economies and how they function. Social Grid Agents (SGA) software is being developed in research at Trinity College Dublin to address these complicated social and economic interactions and also increasing grid complexity due to different middlewares[44]. SGA use a web services interface to a Globus Toolkit4 (GT4) container.

Having moved or copied the tar file to a SGA computer, integration of the Master Process to a web services client would allow secure invocation of methods on a SGA using web services over the https protocol. The SGA in turn would decide which Production Grid Agent (PGA) is most suitable to carry out the work. This option will give the project to flexibility to use the BOINC server as the resource available to one of the PGAs.

Given that the development of Social Grid Agents is still in research, it would be prudent to develop this work in such a way that it can interact directly with BOINC, directly to grid RBs and also have the option to use web services to interface with SGAs.

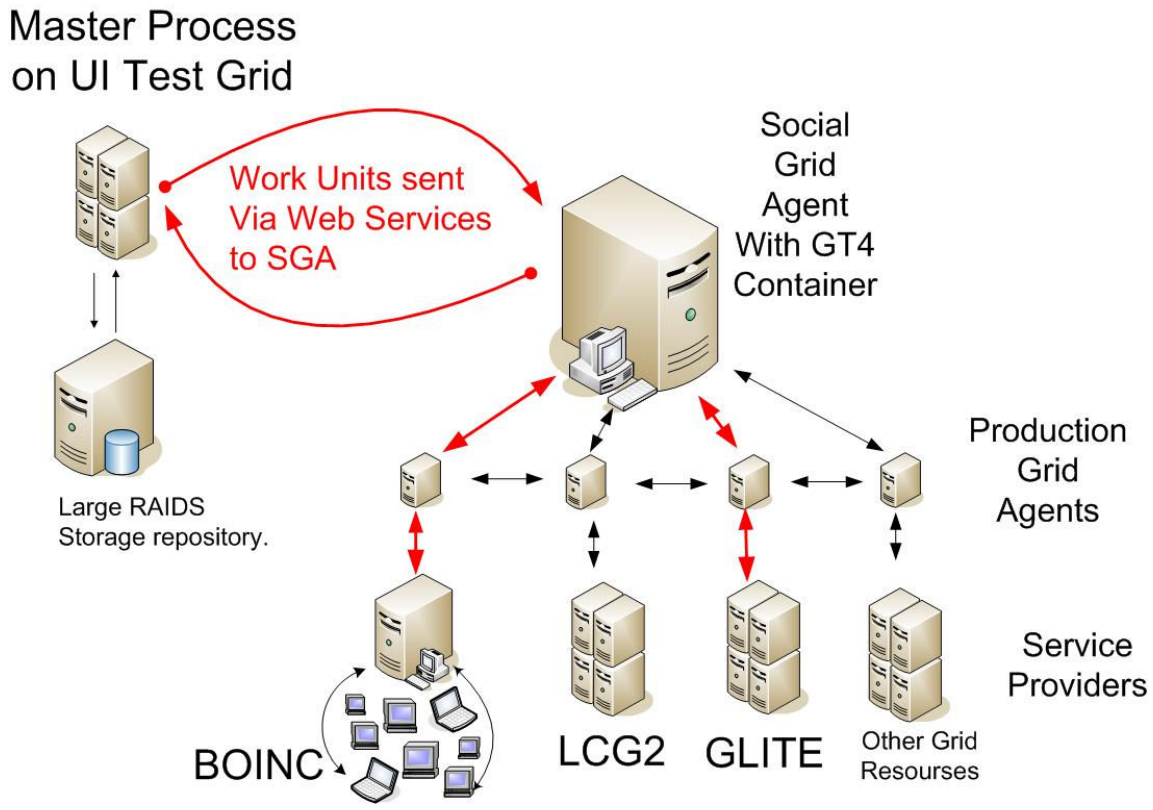


Figure 14 - Master Process interaction with a Grid environment

Section 13.03 Security around Planner Process

Planner and results processes are now on running within the ui.testgrid environment and are temporarily using SSH and SCP with RSA public and private keys set in place. From a grid security point of view, this is not ideal as it could be considered a vulnerable point in the grid infrastructure.

Section 13.04 Flexibility around the size of block files per work unit

In the current Master Process and database design, all block files are of fixed size and contain 100 ligands. Currently, there is no mechanism to change this number either by manually passing in an argument or otherwise. Depending on available resources or the characteristics of the structures to be screened, it may be logical and beneficial to change the number of ligands disseminated with each work unit.

(a) Changes Driven By Receptor Size

For a given receptor, the time taken for completion of a work unit is a partly driven by the receptor file size. An example of this is illustrated when screening a small number of ligands against a number of different size ligands. Various test runs during the design stage (sec 5.10 (a)) showed that multiple ligand runs take on average 8 minutes per ligand screened against a receptor file of 400KB. When using receptor files around 2.0MB in size, for even relatively small ligands this time can increase to 30 minutes per ligand (see table 2). Extending this calculation, a block of 100 ligands screened against a large receptor file would create a job which would require roughly 2 days to screen. This may make the duration of a job infeasible long for some volunteer computer donors. It may also exceed the lifetime of a proxy certificate issued when accessing grid resources.

(b) Changes Driven By Available Resources

A possible implementation of this would be the generation of block files at the instant the work unit is created. The optimised block file size could be driven by the available resources and conditions at these resources (e.g. grid, BOINC, etc.). Some form of querying of conditions from these nodes would be necessary. It would also be desirable to generate custom work units for the various environments to which work units could be sent. Examples of these are jobs sent directly to grid resources, to a SGA or directly to a BOINC server.

(c) Areas of change to Optimise Block file size

As the size of the block file is currently fixed in the implementation of the creation and

storage of the structure files, a change to this feature would involve several areas.

The redesign of the `tbl_<prot_name> _jobs` table would be necessary, potentially allowing each individual ligand to be considered as a job and selecting a number of these to creating the block file of ligands. The challenge of listing all the permutations of receptor and ligand pairs would need to be revisited.

Changes would also be necessary to the *work_unit.py* class in the ProtoPlanner program in the area of file naming. The format and contents of the file name has implications in the result processing classes.

Section 13.05 *Changes to structure of eHiTS for Reusing Pre-process data*

eHiTS spends a large proportion of the time spent on a docking run preparing receptor grid and graph data. Although this data can be reused when docking any ligands against this receptor on the same computer, this effort has to be repeated for each new computer (or worker node) on which the receptor is screened. This is time consuming. This data is stored in the `ehits_work/preprocess` directory. It would be possible to create this data once within the master process as part of the preparation process of a set of receptors for a challenge. The quantity of data produced by the pre-process stage is many multiples in size (e.g. a 400KB receptor produces 20MB of data). Typically, this amount of data is not best suited to transfer over the internet to volunteer computers in people's homes and offices. However, storage capacity and bandwidth within grid infrastructure is designed to work with quantities similar and larger that this. An example of grid storage capacity is NIKHEF[45] with 4 Petabyte storage facility. This is in turn connected to other grid sites and facilities such as Cern, Switzerland via a 10GB connection. Storage and bandwidth such as these make the data requirements within current grid capabilities. Therefore, this optimisation would only be feasible in grid infrastructure where high bandwidth and storage are available. The time taken to download large quantities of data produced by pre-processing may nullify the benefit of a single pre-process step in a volunteer paradigm.

Some changes would have to be made to the eHiTS scripts to allow this optimisation. The automated features of eHiTS are geared towards desktop use for relatively short processing cycles with small number of receptor – ligands pairs. Optimisation and redesign of this control script would allow the program to be passed pre-processed data for receptors and ligands which it could screen with maximum efficiency while minimising redundant or repeated effort. In effect, the C++ libraries can be passed the pre-processed data directly. An optimisation such as this would require further development in the MP, eHiTS, the database and or utilisation of a storage element within grid infrastructure.

Section 13.06 *Speed of Master Process*

At the current implementation stage, the minimum interval between each work unit creation event is set a 30 seconds. This was chosen for testing and clarity purposes. This interval is not short enough to yield a practical and efficient rate of job generation and is acceptable at prototype stage only. Even using block files of 100 ligands and screening against 400 receptors, it would be necessary to generate work units at an interval of roughly 2 seconds. At this frequency, enough work units to screen the 2.02 million ligands from the ZINC database would be dispatched in just over six months.

The main variable and possible impediment to speed for work dissemination is network transfer speed for system calls and file transfer. A multi-threaded or multi-instance version of the ProtoPlanner program with access to sufficient bandwidth would greatly reduce the impact of network speed.

No locking is currently implemented on the database. This means that multiple instances of the ProtoPlanner program could be running and accessing the table `tbl_<prot_name>_jobs` to retrieve the next job data. It is possible that the same job could be retrieved twice before the first accessor of the record has updated the `start_n` field to 1 (giving an unrepeatable read error). Write locking on a record in the records accessed in *`tbl_<prot_name>_jobs`* for the stages between accessing a record and updating the *`start_1`* or *`_2`* field would prevent this.

The database for this work is implemented in MySQL 5.0. This uses the MyISAM database engine which does not facilitate transactional locking. Consideration should e given to this if making the above changes.

Section 13.07 *User interface for Interested Parties*

It is envisaged that a screening challenge of this size will take a number of years to complete. It is also possible to add both more receptors and ligands while the work is ongoing. It would be useful to implement a web interface using html or a similar technology to query the table `tbl_prot_list` and any of the tables in the `prot_<prot_name>_jobs` group to calculate and display the percentage of the challenge complete. This data could also be disseminated to the BOINC server and clients for the benefit of volunteer clients.

Article XIV. Future Work (II)

There are many possibilities for future work with this project. Some aspects of the BOINC project are still to be implemented. These include the project web-page, asymmetric key security, a graphics screensaver, locality scheduling, porting code to more platforms, a PayPal utility, and upgrading the Validator daemon. Further work has to be done to integrate the BOINC project with a Grid. This involves reducing the dependency on system calls, and implementing a Globus communication wrapper. Although, the value of retaining some of the system calls functionality of the current implementation should also be considered. The extent to which related work in the field can influence this project has to be ascertained. The screening software needs to either be improved, or replaced. The publishing of a research paper would increase interest in the FightMalaria@home project.

Section 14.01 FightMalaria@home web-page

Each BOINC project has to have a project homepage. Likewise, a homepage has to be implemented for FightMaralia@home. Potential uses have to be able to negotiate through the BOINC website to the FightMaralia@home homepage, from which they can find out about the goals of the project, find out how to join, and interact with the statistics of the project.

Section 14.02 Project security

For the development of this project, the two sets of public and private keys, that are needed for code signing and file uploads security measures, were generated using a BOINC automatically key generation service. This service is provided with BOINC for testing purposes. The keys that are generated automatically are not secure. To increase project

security both the code signing, and file upload keys must be generated using a crypt_proj function that is provided by BOINC.

Section 14.03 Project screensaver

No screensaver has been implemented for running on the BOINC client machines. A screensaver shows real-time graphics relating to the progress of processing that is currently running a users machine. Such a screensaver would move to not only retain the interest of existing volunteers, but also attract the interest of potential volunteer users. The success of the BOINC-powered side of this project hinges on the willingness of volunteers to donate their processing time to this project, and providing a volunteer with something in return for their donation would go some way towards ensuring the success of the FightMalaria@home project.

Section 14.04 Locality scheduling

Locality scheduling is implemented in BOINC to achieve greater processing and networking efficiency. The facility is provided to restrict BOINC clients, where possible, to processing certain work-units, the input files of which would already exist on their machines due to prior processing of related work-units. This has meaningful implications for the pre-processing computation that has to be carried out on a client for each ligand and receptor that it is about to processed. Should a certain receptor, for example, have been used in a previous screening computation, then the pre-processing stage can be foregone. Locality scheduling can therefore be used to limit the pre-processing that has to be carried out on the clients. It would also reduce the overall number of files that would have to be downloaded to the clients.

Section 14.05 Porting BOINC project to Grid

As it stands, the BOINC project is installed on a machine that is not integrated with the Grid-Ireland Grid. In order for the BOINC project to ultimately integrable with the Grid, the configuration on the current machine has to be mimicked on a Grid machine. This entails re-installing all of the BOINC-dependent software, and transferring the BOINC project code, configuration data, and installation structure onto the Grid machine.

Section 14.06 Change from system call paradigm

At present, work-units are created by using system calls that are invoked from a remote source. Should this BOINC project be integrated into a Grid environment, then the system calls will have to be upgraded to either a RPC functionality, or direct code invocation. The RPC functions could be invoked by a remote resource using Grid communications, and code residing on the BOINC server machine could be invoked locally on the reception of a Grid communication. However for ease of implementation, the development of a C/C++ library for direct invocation would be far simpler than the development of an RPC framework.

Section 14.07 Setup of Globus Toolkit container

A GT4 container is required to receive communications from, or be invoked by, other components in a Grid infrastructure. The GT4 container is also needed to re-invoke, or respond to, the Grid components that may have invoked it. Globus technologies use Web Services with which to communicate. The GT4 container would reside on the BOINC server machine. GT4 containers can be configured to invoke, or interact with, a number of programming languages. For the purpose of this project, C Web Services Core (C WS Core) should be setup with the GT4 container. This means that the GT4 container can receive a Web Services communication, and format the data relating to the message in such

a way that it can be passed as arguments to selected C functions. Likewise, C functions can be invoked in such a way that their arguments can be formatted into a Globus communication by the GT4 container. Files can also be transmitted with Globus communications.

A GT4 toolkit would essentially enable the BOINC project to be assimilated into a Grid environment. The combination of the GT4 container being wrapped around the BOINC server is referred to as a Grid Agent (GA). Processing jobs would be sent from a Resource Broker (RB) to the BOINC GA. The data contained in these processing jobs would be similar to that of the current setup: a zipped file containing both ligand and receptor files, and arguments that are used to create and define a work-unit. Once the work-unit has been processed with BOINC, the results file and related data can be passed back to the calling Grid component in a format similar to that of the initial processing job.

Section 14.08 Research related projects

As mentioned in section 2, some work has already been done in the area of integrating a BOINC project with Grids. This area could be researched to find out if any such implementations could influence or assist in the success of this project. In fact, two BOINC utilities that have been developed at CERN are included with the BOINC software called kill_wu and poll_wu. The applicability of these tools could be investigated.

Section 14.09 Re-evaluate screening software

Further liaising with Symbiosys Inc. is needed to improve the performance of eHiTS on the BOINC clients. The possibility of formatting eHiTS so that its functionality can entirely be represented in a C/C++ shared library could be explored. Having a direct C/C++ interface to screening software might reduce some of the vulnerabilities and inconsistencies of invoking proprietary software of the BOINC clients. Furthermore, the availability of

alternative screening software that can be supplied in this format could be investigated.

Symbiosys Inc. also need to be re-contacted about the time-out error that occurs when eHiTS is invoked on the BOINC client. This error occurs for approximately 50% of work-units, and must be reduced to achieve any sort of worthwhile processing efficiency.

Symbiosys Inc. intend to release a version of eHiTS for Windows. To achieve the maximum benefit of operating a BOINC project, alternative screening should be found should Symbiosys Inc. not produce a Windows eHiTS. Furthermore, locating screening software that runs on MacOS would open up FightMalaria@home to the maximum amount of volunteer users.

Section 14.10 Configure project code for different platforms

eHiTS is currently only available to run in Linux. It was for this reason that the BOINC-independent code for this project has also only been developed to run in Linux. To make the code portable to the three major platforms that are support by BOINC (Linux, Windows, MacOS), pre-processor definitions in the project code would have to be written to include the header files relating to the platform in which the code is being run. Further work would also be needed to release versions of the project executable so that BOINC clients running on different platforms could ensure that the FightMalaria@home code is applicable to their particular platform.

Section 14.11 Increase functionality of Validator

At is stands; the purpose of the Validator is trivial. It is intended by BOINC that the Validator be used to select a canonical result from redundant results. However, as redundancy is not implemented by BOINC in this project, the Validator could be put to more effective use. To increase the efficiency of processing that is sent to the BOINC

project, the Validator could be upgraded to re-issue work-units should they be seen to have failed. The Validator could parse the contents of result files, and in doing so judge whether a work-unit should be re-processed by a BOINC client.

Section 14.12 Consider retaining some aspects of existing implementation

It could be considered wasteful to sacrifice the current implementation, where both the Master Process and the BOINC project successfully function in unison over a remote interface. Further thought could be given to the possibility of having a dual relationship between the Master Process and the BOINC project, in which processing jobs could be sent either directly to BOINC using the current interface, or indirectly by having BOINC, among other Grid resources, anticipating the reception of processing jobs from a Resource Broker using generic Grid communication links. The decision of whether to directly invoke the BOINC project or not might be made on the basis of whether other Grid resources may be heavily utilized at any one time.

Section 14.13 Implement PayPal utility

BOINC provides PayPal functions that can be integrated with a BOINC project to accept donations to further the ongoing success of a project. Such funding could go some way towards achieving a propitious outcome for the goals of this project.

Section 14.14 Publish research paper

Publishing a research paper for bioinformatics conferences could further the success of this project. A published paper might create interest in the project in the bioinformatics field.

Not only might such a paper generate possible funding for the project, but it could also prove to be a means for attracting larger numbers of volunteer users for FightMalaria@home.

eHiTS is normally proprietary software provided by Symbiosys Inc.; however they have configured it to be used in this project for no cost. In order for Symbiosys Inc. to get some benefit from FightMalaria@home, and also to ensure their continued interest in the development of eHiTS for this project, publishing a paper in which they are mentioned would mean that commercial interest could be generated for them in return for their investment.

Article XV. Conclusions (I)

An investigation and review was carried out around biological data and the computer aided docking of these structures. Grid technology and its role in distributed and massively parallel calculations were also explored. In conjunction with inputs and requirements from Dr Anthony Chubb of RSCI, all of the above areas investigated, fed in to a design process which lead to the development of a Master Process which is used to break down the challenge in to manageable parts and dispatch them for processing to the appropriate station. The implementation of the Master Process was successful. Work units were generated from data stored in a scalable database and disseminated to a remote computer in a secure fashion. When results were returned from the processing location (BOINC server in this case), the data returned was processed to persistent storage. Details are outlined of the factors considered in this design process. Particulars and algorithms for the selection of work data are also outlined. The final product is however not ready for large scale deployment but is a basis for further work. Possibilities for future work are outlined at the end of this work in many areas. Many of these are based on experiences gained during the entire process.

Article XVI. Conclusions (II)

This document gives a comprehensive review of the preparation, research, and implementation of the FightMalaria@home BOINC project. The document begins with an introduction to Volunteer and Grid Computing. The related work that has been conducted in the field of BOINC and the integration of BOINC with Grids is then discussed. An overview of the technologies that have been used in the implementation of this project is provided. A comprehensive review is given of BOINC framework, including its architecture, and the tools and API libraries that are to be harnessed to develop a BOINC project. The methods that were used to develop and implement the FightMalaria@home project are then detailed, giving comprehensive specification of the programming logic that has been applied to complete the task of enabling biological screening software to be invoked on BOINC clients. Scope is then given to discuss the problems that were encountered in the development of the project, and the reasons for these problems. A concise critical analysis is made regarding the success of the design and implementation of this project. The success of the project is judged with respect to the original goals that were set out in the project proposal, and the applicability of the architectural and software decisions that were made throughout the lifetime of the project. Finally, a plan of future work is provided that gives an acute synopsis of the prospective work that would bring this project to a successful conclusion.

The implementation of this project has answered the question of whether screening software can be applied to a BOINC project. eHiTS software can run successfully in the FightMalaria@home BOINC project. Furthermore, thought and planning have been applied to designing the architecture of this project with a view to it being integrated into a Grid environment. The architecture is such that compatibility with current Grid infrastructure technologies is possible. The development work that has been carried out as a whole has significantly increased the chance of achieving the goal of screening large drugs databases against the malarial genome in the hope of finding a cure for malaria.

Article XVII. Appendix 1

Example of ligand SD File.

```
ZINC03830813_000000000621
  -OEChem-04270608223D

 15 14  0     0  0  0  0  0  0999 V2000
    0.3999    3.8904   -1.1626 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.2821    3.4528    0.0083 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.6828    3.8678    1.2374 O   0  0  0  0  0  0  0  0  0  0  0  0
    1.4507    1.6376   -0.0030 P   0  0  0  0  0  0  0  0  0  0  0  0
    2.1794    1.2174   -1.2207 O   0  0  0  0  0  0  0  0  0  0  0  0
    2.9293    4.2170   -0.1531 P   0  0  0  0  0  0  0  0  0  0  0  0
    3.5500    3.7872   -1.4260 O   0  0  0  0  0  0  0  0  0  0  0  0
   -0.5850    3.4334   -1.0661 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.2991    4.9757   -1.1558 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.8570    3.5738   -2.1001 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.1891    3.6234    2.0241 H   0  0  0  0  0  0  0  0  0  0  0  0
   -0.0110    0.9628    0.0073 O   0  5  0  0  0  0  0  0  0  0  0  0
    2.2685    1.1664    1.3013 O   0  5  0  0  0  0  0  0  0  0  0  0
    2.7804    5.8201   -0.1431 O   0  5  0  0  0  0  0  0  0  0  0  0
    3.8577    3.7566    1.0791 O   0  5  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  0
  1  8  1  0  0  0  0
  1  9  1  0  0  0  0
  1 10  1  0  0  0  0
  2  3  1  0  0  0  0
  2  4  1  0  0  0  0
  2  6  1  0  0  0  0
  3 11  1  0  0  0  0
  4  5  2  0  0  0  0
  4 12  1  0  0  0  0
  4 13  1  0  0  0  0
  6  7  2  0  0  0  0
  6 14  1  0  0  0  0
  6 15  1  0  0  0  0
M  CHG  4  12  -1  13  -1  14  -1  15  -1
M  END

>  <fm_tagname>
ZINC01683177_000000000621

$$$$
ZINC03831573_000000000844
  -OEChem-04270608223D
```

```
  9  8  0     0  0  0  0  0  0999 V2000
    1.2864    1.5618   -0.0018 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.1449    3.0852    0.0077 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.1768    3.6433    1.6253 Br  0  0  0  0  0  0  0  0  0  0  0  0
    2.9298    3.9092   -0.0049 Br  0  0  0  0  0  0  0  0  0  0  0  0
    0.1462    3.6606   -1.5850 Br  0  0  0  0  0  0  0  0  0  0  0  0
   -0.0110    0.9628    0.0073 O   0  0  0  0  0  0  0  0  0  0  0  0
    1.8401    1.2428    0.8812 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.8231    1.2523   -0.8987 H   0  0  0  0  0  0  0  0  0  0  0  0
    0.0021   -0.0041    0.0020 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  0
  1  6  1  0  0  0  0
  1  7  1  0  0  0  0
  1  8  1  0  0  0  0
  2  3  1  0  0  0  0
  2  4  1  0  0  0  0
  2  5  1  0  0  0  0
  6  9  1  0  0  0  0
M  END

>  <fm_tagname>
ZINC03775300_000000000844

$$$$
```

Article XVIII. Bibliography

[1] Coghlan, B.A., et al "Transactional Grid Deployment" Proc.Cracow Grid Workshop Dec, 2004
[2] The Grid : Blueprint For A New Computing Infrastructure, Kesselman, Foster (1998), ISBN-10: 1558604758
3 Stephen Childs et al. 2006 "A virtual TestGrid, how to replicate a national Grid" Proc. ExpGrid workshop, HPDC 15..
[4] http://en.wikipedia.org/wiki/Volunteer_computing
[5] http://boinc.berkeley.edu/volunteer.php
[6] Volunteer computing: grid or not grid? David Anderson. ISGTW, July 2007.
[7] Enabled Grids fro E-SciencE, (www.eu-egee.org).
[8] EGEE Press release 2005, (http://public.eu-egee.org/files/battles-malaria-grid-wisdom.pdf)
[9] Design of New Plasmepsin Inhibitors: A Virtual High Throughput Screening Approach on the EGEE Grid, Kasam, Zimmermann. (http://pubs.acs.org/cgi-bin/asap.cgi/jcisd8/asap/pdf/ci600451t.pdf)
[10] FlexX high throughput screening (http://www.biosolveit.de/FlexX)
[11] EGEE Press Release (www.eu-egee.org/pr/uk/nr_avianfluegee-eng.pdf)
[12] Sun Press Release 2007, (http://www.sun.com/solutions/documents/solution-sheets/ls_ehits.pdf)
[13] BOINC combined - Credit overview. BOINCstats.com. Willy de Zutter, 2007-01-30.
[14] http://www.top500.org/system/7747
[15] http://www.gravitywell.org/seti/seti@home_radar.gif
[16] http://upload.wikimedia.org/wikipedia/commons/thumb/f/f1/BOINC_screenshot.png/800px-BOINC_screenshot.png
[17] http://en.wikipedia.org/wiki/BOINC
[18] http://en.wikipedia.org/wiki/BOINC_client-server_technology
[19] http://www.malariacontrol.net/
[20] http://www.boinc-australia.net/malaria/
[21] http://africa-at-home.web.cern.ch/africa%2Dat%2Dhome/malariacontrol.html
[22] http://boinc.berkeley.edu/trac/wiki/GridIntegration
[23] Expanding the reach of Grid Computing: Combining Globus and BOINC-based systems. Myers, Bazinet, Cummings. University of Maryland
[24] UnitTest, implemented in Python, based on the program written by Kent Beck and Erich Gamma.
[25] www.rscb.org, www.chemcomp.com.
[26] Daylight Chemical Information Systems, www.daylinht.com/smiles/
[27] http://www.simbiosys.ca/ehits/
[28] AutoDock, Olson Laboratories.
[29] Dell GX270, single 2.5 GHz CPU, 1 GB of memory.
30 Science Direct, eHiTS: A new fast, exhaustive flexible ligand docking system Zsolt Zsoldos , Darryl Reid, Aniko Simon, Sayyed Bashir Sadjad, A. Peter Johnson.
31 Cambridge Crystallographic Data Centre, Cambridge, UK (www.ccdc.cam.ac.uk).
[32] Using Scientific Linux 3.07 Operating System.
[33] Dalby; Nourse; Hounshell; Gushurst; Grier; Description of several chemical structure file formats used by computer programs developed at Molecular Design Limited, Journal of Chemical Information and Computer Sciences, 1992, 32, 244-255.
[34] Definition from California State University, Monterey Bay, (http://it.csumb.edu/departments/data/glossary.html)
[35] http://searchoracle.techtarget.com/
[36] Database Systems: A Practical Approach to Design,Implementation and Management, Connolly, Begg ISBN-10: 0321210255
[37] http://gpu.sourceforge.net
[38] http://boinc.berkely.edu/trac/wiki/DataBase
[39] Eclipse programming framework from IBM, (www.eclipse.org).
[40] Redhat™ Package Manager (www.rpm.org)
[41] EyesOpen Chemical Software, (www.eyesopen.com).
[42] The Royal College of Surgeons, Dublin 2, Ireland.

[43] Details of Grid-Ireland security and certificates (http://www.grid.ie/getting-a-cert.html)
[44] Gabriele Pierantoni, Eamonn Kenny, and Brian Coghlan. An Architecture Based on a Social-Economic Approach for Flexible Grid Resource Allocation. Cracow, Poland, November 2005.
[45] National Institute for Nuclear Physics and High Energy Physics, Amsterdam. (www.nikhef.nl, www.dutchgrid.nl)