FYP II: FINAL REPORT

Implementation and Evaluation of a Thermal-Aware Campus Grid

by

Guilherme Dinis Chaliane Junior
14163

Bachelor of Technology (Hons)
(Information Communication Technology)

January 2014

Universiti Teknologi PETRONAS
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

Implementation and Evaluation of a Thermal-Aware Campus Grid

by
Guilherme Dinis Chaliane Junior
14163

A project dissertation submitted to the

Information Technology Programme

Universiti Teknologi PETRONAS

in partial fulfillment of the requirement for the

Bachelor of Technology (Hons)

(Information and Communication Technology)

Approved by,

_____

(Dr. M Nordin Zakaria)

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

January 2014

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein has not been undertaken or done by unspecified sources or persons.

_____

Guilherme Dinis Chaliane Junior

ACKNOWLEDGMENT

First and foremost, I am thankful to the creator, for blessing me with life. I am eternally grateful to my mother, Olinda da Luz, for the years of guidance, care and friendship. Your life and wisdom have always been an inspiration for me to pursue my dreams, and become the best person I can be. To my sisters, Clavia, Telma and Nicole, thank you for your loving support over the years.

I would also like to express my gratitude to everyone who was involved in this project, starting with my supervisor, Dr. M Nordin Zakaria, whose mentorship has always steered me in the right direction, and made me strive for excellence. To the people at the laboratory, I give my deepest gratitude for their aid, and shared knowledge.

Last, but not least, I would like to thank my sponsor, PETRONAS, for funding my studies over the past 4 years. It has been a once in a lifetime experience.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1 ABSTRACT

Volunteer and grid computing framework systems have helped create several of today's largest resource pools for scientific and engineering computing. At the same time, as computers evolve, there is a greater demand for more environmental-aware scheduling systems to be deployed with these frameworks in order to address existing concerns of the preservation of resources like energy, and computers themselves. In response to this concern, the High Performance Computing Services Center of Universiti Teknologi PETRONAS (HPC-UTP) sought to incorporate thermal-aware scheduling into its campus grid. To achieve this, the center decided to modify an existing Volunteer Computing (VC) framework to make use of different schedulers at run time, without the need to recompile the server. This thermal-aware, dynamic-scheduling capable framework will be deployed on a test environment, to assess its viability for the university's campus grid.

2 BACKGROUND

Grid computing can be loosely defined as the computation done by a group of interconnected computing devices, generally computers, trying to achieve a common objective. In and by themselves, grids are just networked computers. However, when seen as whole, they provide large scales of computational and data storage capacity to solve complex problems. Grids are one of the basic forms of distributed computing taxonomies, along with clouds. In modern society, cloud computing systems have been gaining wide attention from both IT service providers and users; though when seen from an infrastructure point of view, clouds can be thought of as a set of interconnected grids and other devices with levels of abstractions in their communication and coordination.

To deploy a grid computing infrastructure, one needs to make use of software systems that can establish the connection between devices, for the purpose of enabling process coordination and data exchange. Such systems can be thought of as middleware, because they often create a level of abstraction between what can be homogenous or heterogeneous devices.

Grids are not always formed by pools of dedicated computing resources. In fact, since the early 1990s, the largest grid deployments worldwide have been built on resources donated by the public, in the form of spare computational resources shared during computer idle periods. Projects like SETI@Home, which searchers for extraterrestrial life, were among the first grid projects ever created that worked with this paradigm, which later became known as Volunteer Computing. Other projects that came afterwards, followed a similar paradigm, and went on to create some of the world's largest open computing systems. With the evolution of these systems some institutions, like the Berkely Open Space Laboratory, began creating grid frameworks to cater to the VC paradigm.

Today, there are several VC computing frameworks in existence. Currently, the Berkeley Open Infrastructure for Networking and Computing (BOINC) can be easily considered the most widely used VC framework in academia. BOINC is also applied in several other grid environments, where resources are either fully or semi-dedicated.

BOINC, just as most of the other existing grid computing frameworks has its own scheduling policies embedded into it. This system is responsible for the distribution of jobs to all the connected devices, from a central server. Consequently the integration of a different scheduling policy into any such system requires substantial rework of the application's code to adjust its working structure to the user's intended needs. As several data and computing centers today try to address the energy crisis of the century, by incorporating smarter algorithms to manage their simulation/computational jobs, the demand for systems that are capable of accommodating different scheduling policies at different times is forthcoming.

The need to incorporate different environment-aware scheduling systems at run-time within an existing VC system that has its own scheduling algorithm, without altering its working framework was the key driver to this project.

3 PROBLEM STATEMENT

Environment-aware scheduling policies are policies that evaluate non-system related aspects of the environment under which machines operate to decide on the best task or data assignment scheme. For example, a heat-aware or thermal-aware system would consider the temperature of a device in scheduling task processing for it.

If one were to integrate a thermal-aware policy into a system such as BOINC, one would surely have to alter the inner structure of the application to accommodate a different scheduling policy. However, at present, there are several thermal-aware scheduling algorithms in existence, and further being researched at HPC-UTP.

Since the usage of any environment-aware scheduling policy generally requires the introduction of sensor-based information into the scheduling policies, the first challenge lies in designing a system that is capable of receiving and processing information from sensor systems. The second challenge lies in providing this information, along with job information as well as any other information concerning the resources of each computer system involved in the grid to the possibly different scheduling systems, for them to carry out scheduling without disrupting BOINC's remaining inner framework.

In summary, the main challenges of the project were to extend BOINC to work with thermal sensor systems, as well as to provide researchers with a simplified and intuitive mechanism to manage different scheduling applications.

4 OBJECTIVES

This project aimed at achieving two objectives:

1. To extend BOINC to work with thermal sensor systems;
2. To deploy a pilot thermal-aware BOINC system on the university's campus

Despite the focus given to thermal-scheduling, this project's scope does not involve the development or implementation of any rigorous thermal algorithms. The main emphasis is on the infrastructure side of the equation, i.e. the modification of an existing framework, i.e. BOINC, and its implementation on pilot project to support thermal-scheduling.

CHAPTER 2
LITERATURE REVIEW

5 LITERATURE REVIEW

Despite the advent of Cloud computing, Grid computing still remains one of the leading forms of managing large scale scientific or engineering simulations and computing projects. Major university and corporate computing projects still rely on the several grid computing frameworks that are available, to harvest the computational power they need. Examples of these include SETI@Home, which is currently the largest distributed computing effort and seeks to find extraterrestrial life [1]; Folding@Home, which seeks to cure diseases like Cancer, and Parkinson's disease by running protein folding simulations [2]; and LHC@Home which runs high-energy particle collisions simulations for the Large Hadron Collider (LHC) at the European Organization for Nuclear Research **(**CERN), the world's largest particle physics laboratory [3].

The fast growing capabilities of today's computer systems require substantial efforts of sustainable management of their resources in order to respond to society's needs of tomorrow. This management seeks not only to expedite data processing, but to do so without the incursion of extra costs to society in the form of greater energy consumption, or that of any other resource for that matter. To respond to this need, the concept of green scheduling has thus emerged in the realm of computing. IT systems have always placed great emphasis on efficiency: doing more within a shorter period of time; however, the objective had always generally been to relief resources so that other computing could take place. Green scheduling, as defined in [4] and [5], aims at reducing energy consumption, especially in cloud services, grid or data centers. Most of the algorithms in it work with sensor technology to monitor energy consumption and adjust work distribution accordingly, thus reducing the consumption of unnecessary resources.

BOINC is one of the leading frameworks used in VC projects, where the nature of the eco-system is one where users freely donate the idle time of their devices (e.g. PCs, laptops, mobile devices, and entertainment consoles) to help solve large scale scientific problems such finding a cure for diseases like Alzheimer's [6].

As presented in [7], BOINC encompasses several key design concepts that make the gathering and retention of millions of personal computer resources donated by the general public on a daily basis not only possible, but feasible and to some extent effortless for users. These include management of trust between scientists and volunteers, the rewarding of users for the CPU cycles donated, the scaling of a project's deployment, and others.

Now, despite having been originally designed to be a VC system, BOINC can actually work as an alternative to a grid middleware system in an environment where a strict grid-oriented system would not be suitable. An example of such an environment is a university campus grid that is based on computers located in academic laboratories, making use of their idle periods to perform computations. The Universiti Teknologi PETRONAS (UTP) has an environment alike.

With over 800 machines distributed in more than 30 labs, UTP's campus grid has the potential to deliver Teraflops of computational power. In order to harvest that potential, HPC-UTP makes use of BOINC. The robustness of the framework, coupled with its open-source nature gives the center the ability to leverage the computers in the university's laboratories at a low cost. In spite of this, there is one issue that concerns environmental preservation of hardware, and that is the constant availability of dedicated machine cooling across all labs. Since air conditioning is only made available during regular office hours, generally between 7am-7pm on weekdays, the machines become dependent on the environment's temperature to establish their own during non-office periods [8]. This can expose the machines to a high risk of damage, if they perform heavy computations during such periods as the indoor temperature can rise above 28°C, as shown in Figure 1.

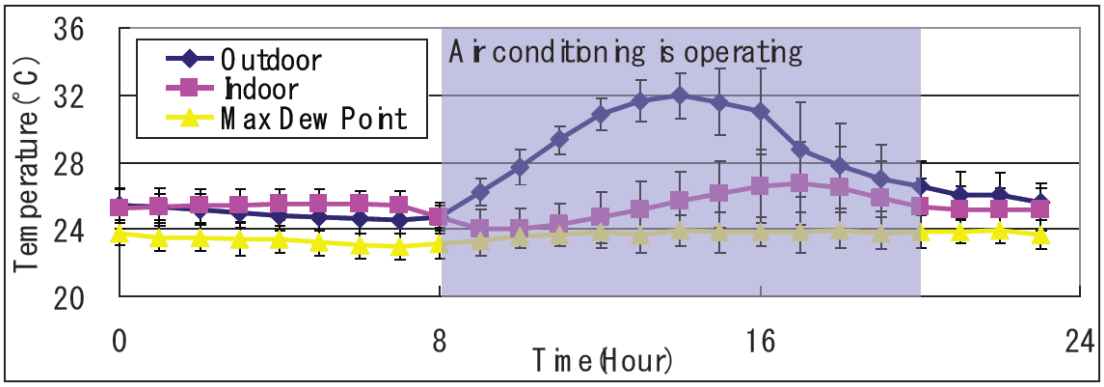


Figure 1 One day transition of temperature and dew point in Ipoh, Malaysia (*adapted from [8]*)

BOINC's resource matching based scheduling policies are capable of achieving high project throughput. By assigning jobs to the first available candidates that can fulfill their requirements, BOINC ensures projects' overall progression, as described in [9]. Nonetheless, there are several different aspects of distributed computing that require different approaches to resource scheduling that the traditional CPU centric batch schedulers do not account for.

In [10], the authors point out that one of the overheads of distributed systems is data transfer. In their work, they presented a data-aware scheduling algorithm for computing grids. Their system managed to reduce data transfer requirements between systems, lowering bandwidth usage and overall jobs runtimes, by assignment tasks based on hosts' proximity to the data source of the files needed by the task. The authors in [11] developed an energy aware scheduling system for data centers that maximized solar energy consumption up to 117%, by scheduling jobs based on predicted availability in the near future and made use of grid energy to avoid deadlines at the times when it would be cheapest, effectively reducing grid energy consumption by up to 39%. In [4], the authors presented a similar green scheduling concept; however, their solution reduced power consumption by overlapping complementary jobs, i.e. CPU oriented with I/O oriented jobs, with a slightly higher level of relaxation compared to an existing Fair Scheduler. Their green scheduler showed 7-9% better energy efficiency. In the center's ongoing work with thermal based scheduling, several algorithms that can schedule jobs according to predicted thermal availability of host machines in a non-dedicated environment, based on sensor data, have been proposed.

Thermal scheduling allows the grid system to allocate work to machines which are deemed thermally viable only; i.e. machines that are expected to be under an acceptable temperature range during the computation, thus reducing the risk of any damage caused by overheat [12][13][14]. The goal of thermal scheduling, as explained by [15] and [16] is to either maximize the workload a machine can have while under an acceptable temperature, or minimize the maximum temperature a machine reaches while executing the maximum number of jobs it can.

The policy used in [11] required sensor based technology to feed the scheduler relevant energy consumption information from host machines, just as the thermal algorithms being research do. This might not be easy to directly apply in a standard VC environment. Nonetheless, if battery performance of portable devices were to be considered, instead of solar power or thermal availability, the concept could be easily mapped. [10] and [17]'s solutions, on the other hand, were based on task data usage prediction and machine's proximity to data hosts as well as bandwidth availability, respectively. Both could be applied in existing BOINC VC projects such as SETI@home or Einstein@home. What these policies show is that, both in dedicated and non-dedicated environments, distributed computing middleware systems should be prepared to address different scheduling schemes. It is the center's belief that the capacity for a middleware system to support these different schemes should be as dynamic as possible so that a single middleware system could be fit for different environments, requiring only an additional or external scheduling application to be attached to it at run time.

To integrate these algorithms on a campus grid, the middleware application in use, BOINC, has to be modified to dynamically make use of different scheduling programs. The goal is to deliver a scheduler-independent, thermal-aware BOINC system, and to implement it in a pilot test environment.

CHAPTER 3
METHODOLOGY & PROJECT WORK

6 METHODOLOGY

To deliver the next stage of the all-encompassing goal of creating and fully dynamic VC and grid computing framework that supports sensor-based environment-aware scheduling algorithms, the author took a tiered-approached to the problem. There were 2 distinct tiers in consideration: system back-end, and the client application.

The first tier, system back-end, concerns the modification of the BOINC server system components responsible managing hosts/client's data, and scheduling available jobs. This portion of the system was modified to accept thermal data from clients, and to assign job scheduling responsibilities to third-party/external (XS) applications, as specified by system administrators and/or researchers.

The second tier comprises of the client application. Most modification efforts here covered the alteration of the client to integrate itself with a thermal sensor system, and to send thermal information to the BOINC server for it to carry out thermal-aware scheduling when applicable.

The strategy used to complete this task was to take a disciplined approach to system development, applying formal project management strategies where they fitted. The project's work breakdown structure (WBS) on Appendix 1 outlines the project timeline, milestones, as well as tasks executed throughout the project's life cycle, from conceptualization to deployment and evaluation.

To provide a working solution within the given time frame for the project's completion, while maintaining a highly flexible work dynamic that supports rapid adjustments, the project followed a throw-away prototyping development approach, shown in Figure 2.

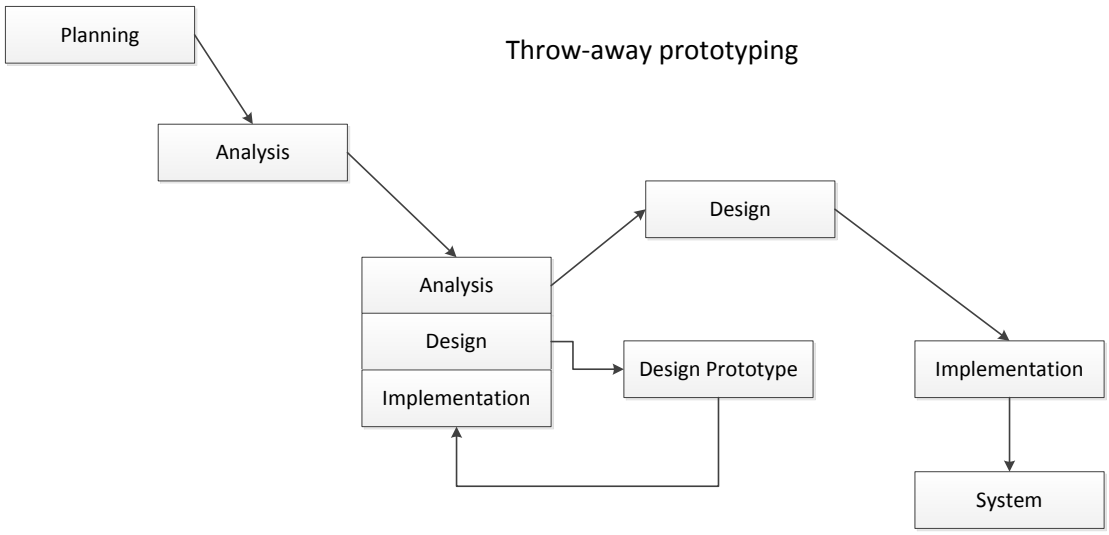


Figure 2 Throw-away prototyping

The system's proposed network deployment architecture is shown in Figure 3.

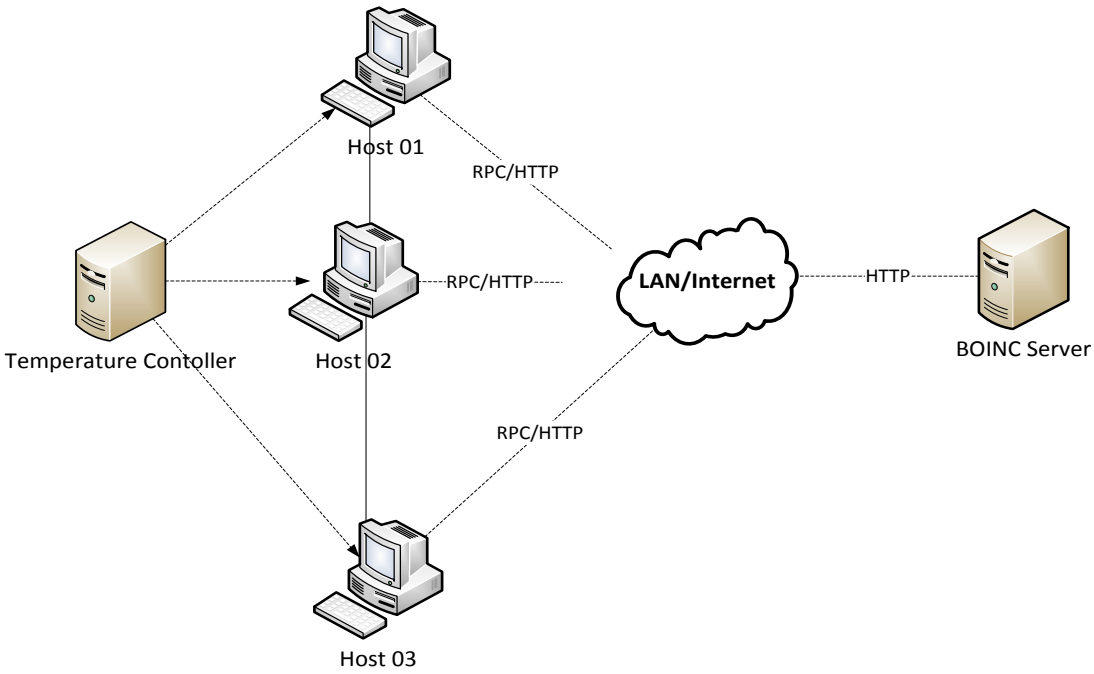


Figure 3 Proposed network deployment architecture

A temperature controller system was setup in the test environment laboratory to regularly monitor room temperature. The hosts in each lab collected the environment's temperature information from this system, and used it while communication with the BOINC server when they made work requests.

7 PROJECT WORK

7.1 The Architecture

BOINC's system uses a client server architectural style. Servers host applications and work-units, and wait for clients to request work from them. The clients are the ones who initiate communication with the server to either report work in progress, or request for new work. BOINC's general component architecture is depicted in Figure 4.

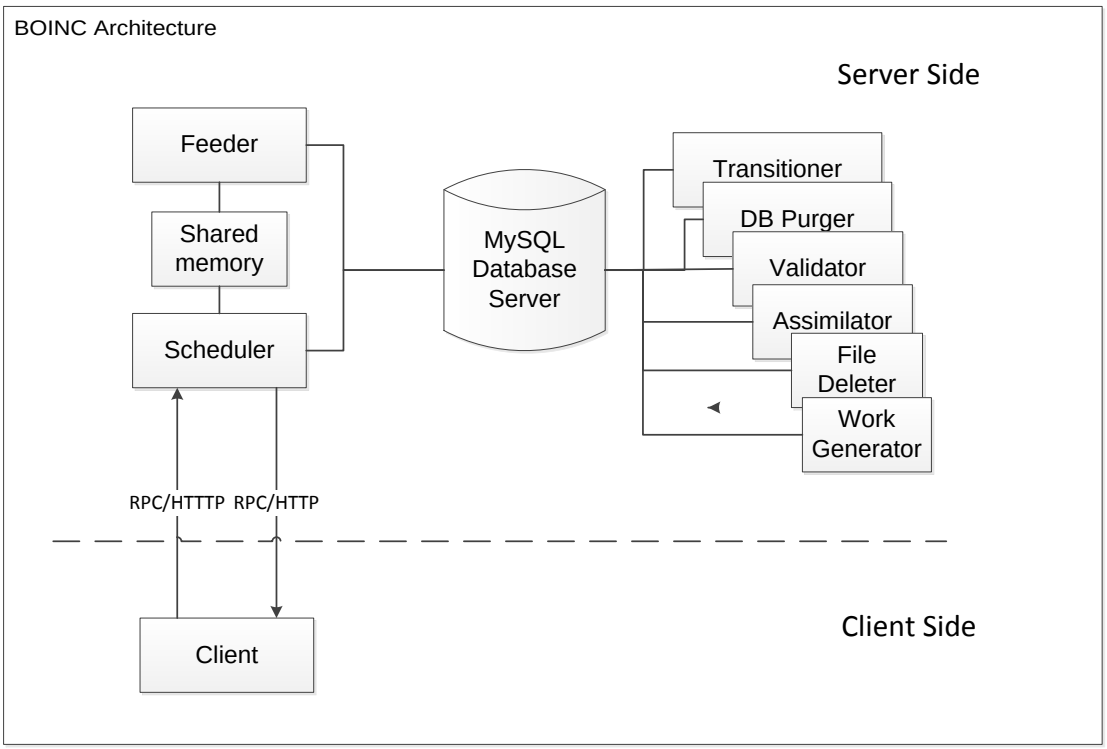


Figure 4 BOINC architecture

All communication between client and server is done via RPC/HTTP requests. When a server receives a request from a client, its scheduler module is the only component that interacts with that request. Meanwhile, the other server components perform tasks to keep the project running in the background. The transitioner manages the work unit/results that reside in the database according to their state. The validator and assimilator daemons are application specific; one verifies results while the other processes the canonical result, handling the output as specified, respectively. The file deleter removes upload and download files that are no longer required by workunits

and results, and the DB purger removes unnecessary database records. The work generator is used to create work units, and finally the feeder and scheduler are responsible for work distribution and result processing. These last two components were the most relevant to this project.

The feeder and scheduler communicate via a shared memory scheme that keeps jobs taken from the database, waiting to be sent out to hosts. When a host sends a request, in the form of an XML file, the scheduler processes it by performing non-trivial user and host validation, and then checking if there are any jobs available, first on the shared memory and, if necessary, on the database, provided that the host has requested work. If the host is simply reporting work in progress, the scheduler responds by notifying the host of whether it should continue running the tasks it has or abort them immediately for a given reason. The reply is sent out as an XML file as well. Once a request is processed, the scheduler updates the database records of the host and the tasks that it's currently running so that the other daemons can process credit attribution and update overall project status

Bearing this working framework in mind, the objective of extending the BOINC system to integrate dynamic thermal scheduling should not interfere with the tasks of the other also important components of the system.

To achieve this, the efforts of the project concentrated on modifying the existing scheduler component to build into it a module that handles third party/external schedulers, i.e. an external scheduler manager (XSM). To schedule job assignment to clients, the external scheduling application has to have both client information, and information about all existing workunits on the server. Consequently, a standardized communication method between the XS and the XSM had to be devised. The chosen approach was to use a standardized XML representation for both information sent from the XSM to the scheduler, and the assignments made by the XS to the server (Appendix 2). This standardized communication format is similar to that used by client-server communication in BOINC. The architecture of the modified system is depicted in Figure 5, and the corresponding deployment diagram in Appendix 3.

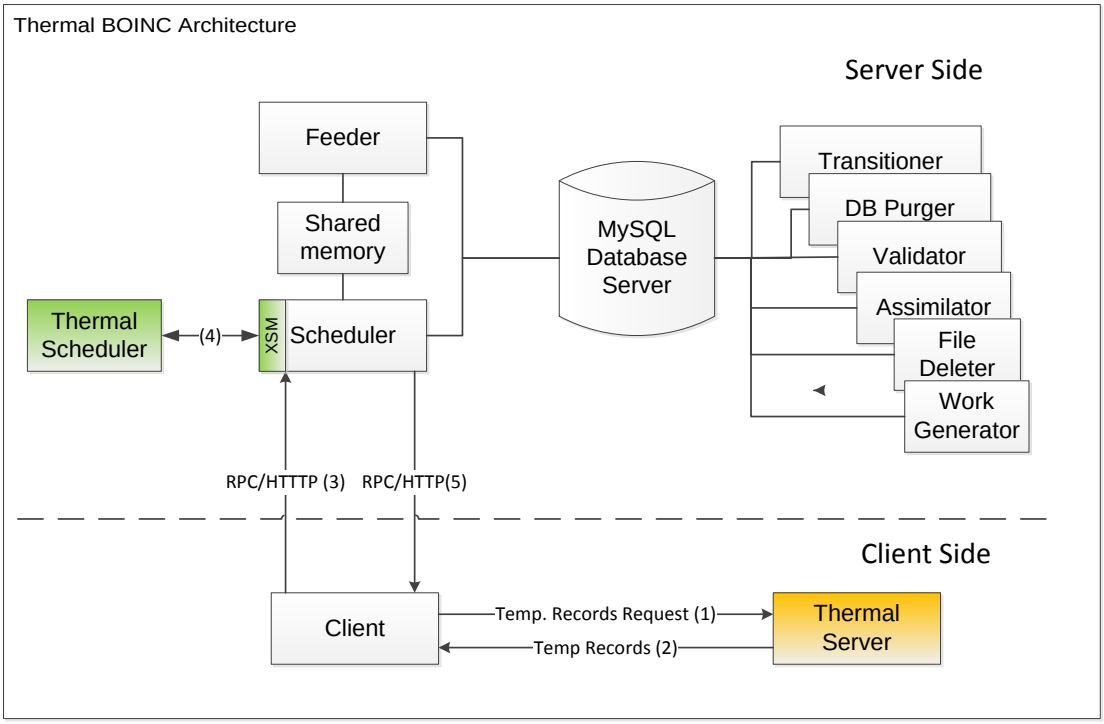


Figure 5 Thermal-aware BOINC architecture

7.2 Test Deployment

Before deploying the system on campus, a proof of concept test environment was be used to run a small deployment of the entire framework. On this deployment, all of the impending components of the to-be system were tested: the thermal server, the thermal-aware client, and the thermal-aware scheduling application.

The environment was being setup in a lab, using commodity desktop computers, similar to the ones used in the university's laboratories, in a closed circuit network.

7.3 Thermal Server

One of the main components of this entire system is the thermal server. The thermal server monitors room temperature, and provides this information to all hosts that request it in its environment.

To deploy this server, the author made use of USB sensor device that could be directly plugged into any computer, and a custom built system to store the sensor's

readings on a public directory which could be accessed via HTTP requests. The configuration is shown in Figure 6.

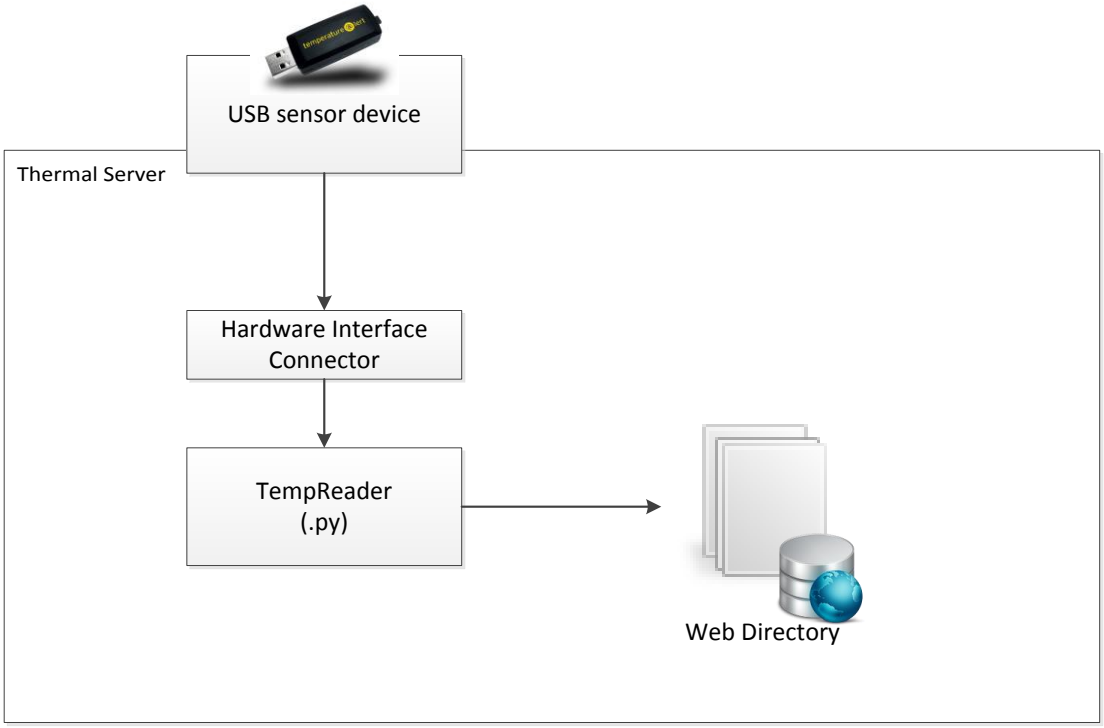


Figure 6 Thermal server configuration

7.4 Client

BOINC's client application was being altered to actively pull thermal data from a thermal server in its environment, and add this information to its work requests. Whenever the client sends a request to the server, it sends information about its hardware specification, software environment and workunits' status. The new client needs only to add to this request the thermal information it pulls from its localized thermal server. Once the server receives this information it provides it to the thermal aware scheduling application, which decides how to allocate workunits to it.

7.5 Dynamic Scheduling Framework

The concept of dynamic scheduling was being developed with the goal of allowing researchers to actively switch between scheduling applications, be them thermal or not, at run time, without recompiling the server system. The idea for the implementation is rather simple: the decision of which scheduling application to use should be configurable by users, and checked by the BOINC scheduler/XSM at each BOINC server startup (or restart).

After reverse engineering BOINC's scheduler system, the author could identify an approach to making modifications that would accommodate the new scheduling paradigm. First though, there were key issues to be taken into consideration:

- **Deadlock**: the implementation had to avoid deadlocks in case of race conditions. Currently, BOINC's scheduler is deadlock free, and multiple scheduling instances can run at the same time for different hosts. If lists were to be used to queue requests for external scheduling applications, deadlock would not have been as easy to avoid.
- **Functionality**: hosts must have their work requests effectively processed. A host could be delayed endlessly waiting for new work to be assigned by an external scheduling application or to be notified of the lack of available work-units.
- **Efficiency**: Reported work should be acknowledged in a timely manner and new work assigned in the same fashion.

Atop of these issues, certain constraints were chosen by the author to be observed while modifying with the system:

- Maximize the usage of existing infrastructure, framework (processes) and code base
- Minimize the level of changes to the original source code.

After a careful, analytical evaluation of the application, the following algorithm was extracted from the existing scheduling system:

```
1. Authenticate users
2. Handle results (work in progress)
3. Resend lost work
4. Abort results
5. Send new work
6. End
```

Figure 7 Boinc's scheduler program algorithm

The modified system, which embedded dynamic scheduling into the framework seamlessly, is given below:

```
1. Authenticate users
2. Handle results (work in progress)
3. Resend lost work
4. Abort results
5. if (configured to use XS scheduler)
     run XSM
   else
     run native scheduler
6. End
```

Figure 8 Dynamic BOINC's scheduler algorithm

The workflow in Figure 8 provides, in theory, two key advantages. First, by making the external scheduler integration (XSI) framework connection at the point of new work distribution only (step 5), the rest of the BOINC system's framework is left intact to carry out its other non-trivial operations as intended: verify users, register new hosts, validate results, etc. Second, by making the call to use the XS optional, users have the alternative to simply use the BOINC as if it were in its original state, which might be necessary or even required in certain cases. Both of these aspects were seen as fundamentally required in maintaining the functionality and efficiency factors described earlier. Deadlock prevention on the other hand, is an issue to be mitigated at the implementation level, and the algorithm designed to address it is show in Figure 9. The algorithm describes how the process flow changes when the XSM runs, using an XS, and when the native scheduler runs instead. Essentially, only

one request can make the scheduler run, either because the run interval or number of hosts on queue limit has been reached; both parameters are configurable. When the XSM runs, the waiting queue is locked, preventing any forthcoming requests from accessing. These requests are given a short back-off period of 10 to 60 seconds to request for work again. If the XS is called, the ready queue is then locked and updated by the same request. This 2-phase lock system precludes conflicting access to the waiting and ready queue, thus effectively preventing deadlock.

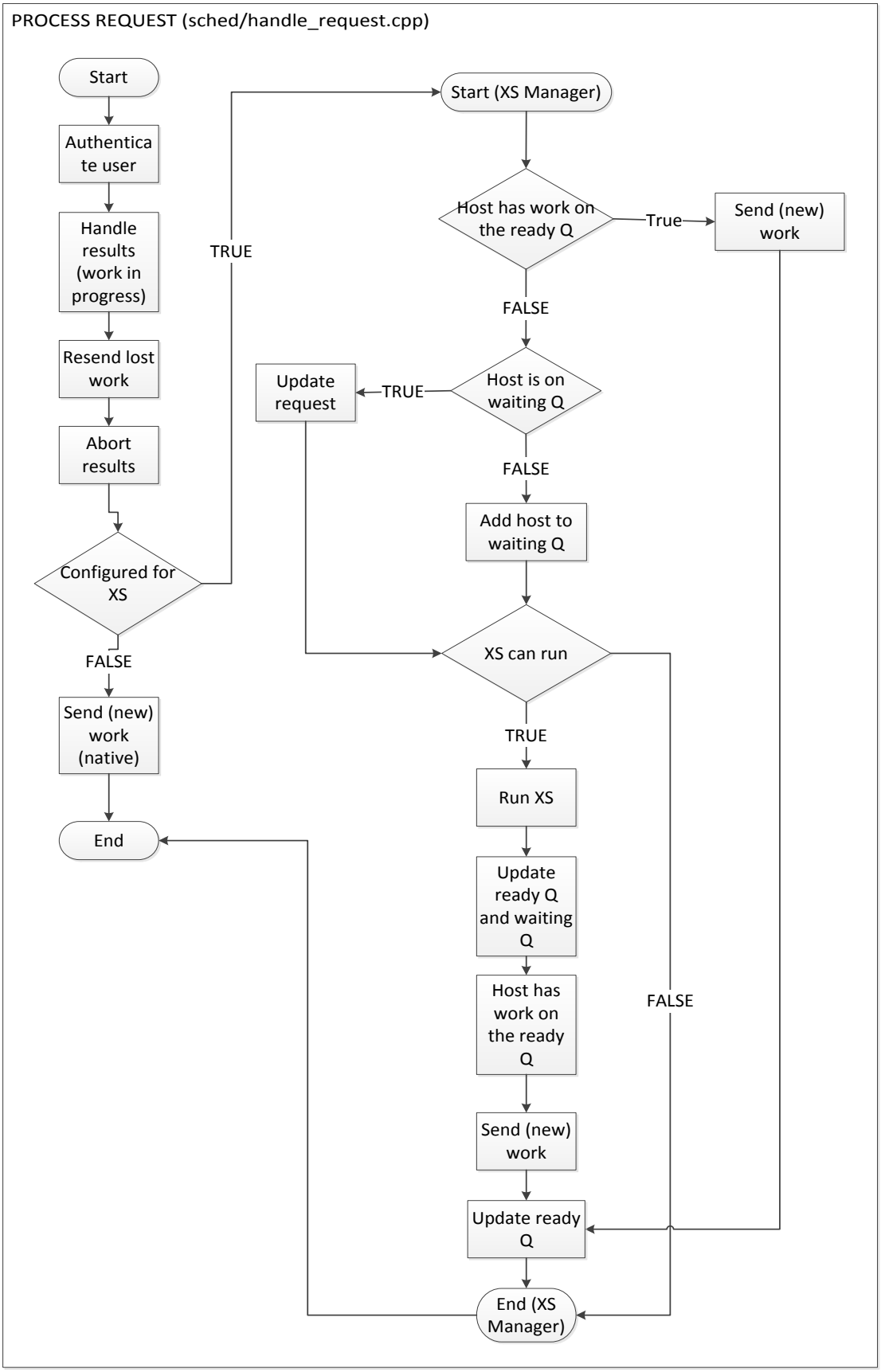


Figure 9 BOINC XSI

CHAPTER 4
RESULTS AND DISCUSSION

8 RESULTS AND DISCUSSION

The purpose of this project was to address one the concerns in the deployment of a computing grid on campus: the environmentally friendly usage of hosts in the grid during non-viable periods. The development of the project was divided in the following stages:

1. Thermal Server
2. Client of Windows
3. Sample Application & Work Generation
4. Server System
5. External Scheduler (XS)
6. Dashboard
7. Thermal Scheduling
8. Mass Deployment

These stages affect certain components of the overall system. We now discuss the process, challenges and current status of each stage.

8.1 Thermal Server

This stage affects the server application that feeds thermal data to hosts in its environment. In it, the development of 2 applications took place: a hardware interface that reads room temperature from a sensor, and a "wrapper" application that uses the first to log temperature and hosts availability to a web directory.

The first application, the sensor reader, was developed using an API provided by the sensor manufacturer, Temperature Alert (http://www.temperaturealert.com/). The application reads temperature from a USB sensor attached to the computer where the program is running. It can provide the readings in either degrees Celsius or Fahrenheit.

The second application is the main program researchers will interact with. Its parameters are configurable, so as to let users tweak its run in a manner that suites their needs. The following are the parameters users can adjust:

config.json

```
{

        "readInterval":60,
        "logFile":"tempread.log",
        "availabilityFile":"/var/www/tempread_ws/availability.log",
        "temperatureFile":"/var/www/tempread_ws/temperature.log",
        "tagFile":"/var/www/tempread_ws/tag.log",
        "maxTAGRecords":"30",
        "thresholdTemp":"29"
}
```

- **readInterval**: defines the frequency with which the temperature sensor should be read, and it is defined in seconds. 60 means the sensor is read every minute (i.e.: at intervals that are 60 seconds apart).
- **logFile**: defines the logging file for the application, used to log errors
- **availabilityFile**: destination of host availability measures after each temperature reading
- **temperatureFile**: destination of the temperature log read from the sensor
- **tagFile**: this parameter is specific for the project. It defines the file where the recent availability of the hosts is stored, to be used by the thermal scheduler, as opposed to all availability readings.
- **maxTAGRecords**: defines the number of recent records to be logged in the tagFile for the thermal aware grid (TAG) project's thermal scheduler.
- **thersholdTemp**: specifies the threshold temperature, used to determine hosts' availability.

Industry guides recommend that server or data room temperature should be set between 26-30° Celsius [18] [19] [20], with some advising against reaching the 27-28° Celsius limit [19]. Currently, the threshold temperature has been set to 29° Celsius because the highest temperature observed in the test environment, functioning under constant cooling, was of 31.56° C, logged on March 11 at 6.59PM while the lowest was 25.06° C registered on March 18 at 9.17PM. A temperature reading above the threshold temperature makes the application set the availability flag in the

availability log to false, i.e. the value of zero. Sample log files can be seen on Appendix 4.

8.2 Client for Windows

The outcome of this stage was to have a compiled client, ready to run on test machines for simulation purposes. To deploy the test project, TAG, the BOINC client application had to be capable of pulling thermal records from a thermal server and appending those records to its XML requests. Since most lab computers run on Windows operating system, a Windows build of the client was modified and tested. During this stage, 2 issues were encountered: thermal data readings access, and size.

The first issue, thermal readings access, had to do with the facilities for running HTTP/GET requests on Windows. BOINC has a set of classes that perform CGI based requests to servers. However, there was no simple mechanism for making HTTP /GET requests to retrieve data files. To circumvent this issue, the author made use of a third party tool called Wget for Windows, a free program that performs HTTP/GET requests. Currently, the modified BOINC client requires this application to be installed on the client machine to be able to download thermal records from the server.

The second issue had to do with the volumes of data the host and server have to handle when managing thermal information. If a host continuously pulls its entire historical availability records from the thermal server prior to sending XML requests to the server, the size of the XML data transferred over the network increases quickly, depending on the frequency of thermal readings. This may not be a big issue for each individual host, but it poses a problem for a server handling multiple hosts' requests on a waiting queue, and to the network traffic. To counter this issue in this project, the records retrieved by the hosts were limited to past half hour, i.e. last 30 records with a read interval of 60 seconds, prior to any request being made. The reasoning for this decision is given in section 8.5, Thermal Scheduler.

The modified client was tested with a native BOINC server prior to being testing with a modified server. Test cases and test results can be found in Appendix 5.

8.3 Sample Application & Work Generation

BOINC has been used to distribute work units from several scientific fields across the world, some requiring user intervention while most do not. For the purpose of this project, the prime directive was to evaluate the system's extensibility in alternating between scheduling applications at run time, and supporting sensor based thermal scheduling frameworks. Thus, the context of the test application was deemed to be less relevant compared to its required work cycles and time to completion.

The test application used was based on one of the sample applications provided by BOINC, named uppercase. It simply converts letters of the alphabet from lowercase to their uppercase counterpart, where applicable. However, a few modifications had to be made to the original application to extend its runtime on the test machines, and to make it more CPU intensive. In defining the estimated work cycles (FLOPS) an application requires, one has to run tests on a machine and derive the value from the runtime and work cycle (FLOPS) capacity of the test machine. The resulting test application had runtime of 10 ~ 15 minutes of the test machines.

A work generating engine, **UppercaseWG**, was devised to create work units and results (jobs) periodically, with randomly generated input, for the test application.

The engine generated random input data files, and created results to be sent to hosts, at periods specified by the user. In the testing environment, where 3 nodes were used, 10 results were generated at regular intervals per hour, i.e. every 12 minutes. If each host took the upper limit of 15 minutes to complete each result, it should be capable of handling 3 to 4 results per hour under the best circumstances. With 3 hosts, the total number of results that could be processed per hour, under the best conditions, was 15 to 18. The gap was there to account for periods of unavailability the hosts could have gone through after being assigned a result. The work generation frequency of the engine can be altered throughout the testing cycle, based on observed behavior.

8.4 Server System

The new BOINC server's main feature was its ability to support different schedulers, switching between them at a run time's notice, and to support thermal data processing. This stage centered on altering the workflow of the native scheduler

component, and that of its companion application, the feeder, as well as the management of thermal information provided by hosts to feed it to external scheduling applications. The latter had already been achieved during the first semester.

Both the native scheduler and the feeder coordinate their work activities via status flags set on results that reside on the work list/queue populated by the feeder. When a new result is added, its status is set to "present". When the scheduler assigns it to a host, the flag is set to "empty" for the feeder to remove it and populate it with a new result. In the modified system, these flags are also being used by the external scheduler manager to notify the native scheduler of the hosts to which the results have been assigned to. Consequently, the interpretation of the flag status depends on whether or not an XS is being used, instead of the native scheduler, especially after a change in the scheduler. This stage ended with a beta build of the server, which is was used in the test environment.

8.5 External Scheduler

In order to assess the effectiveness of the modifications made to both client and server, a scheduling application was built and attached to the server for the scheduling of workunit results created by the UppercaseWG engine, to hosts in the test environment. In building this application, there were certain factors that were carefully considered.

Scheduling Factors

There are 3 factors that were used in judging the work assignment for hosts in the scheduling application: **work request**, **fairness**, and **reliability**.

Work request simply addresses the question of how much work a host is requesting. In doing work assignment, the hosts' capacity has to be compared against the estimated work cycle requirements of the results/jobs to be assigned to it. Each host makes a request for work in the form of CPU seconds. The results have their work cycle requirements expressed in FLOPS. To estimate incremental work assigned to a host, given a result/job in seconds, we use the following formula:

$$IWA = \frac{FPOPS(wu)}{FPOPS(host)}$$

Where:

IWA: incremental work assigned by the workunit's result/job (seconds)

FPOPS(wu): work cycles required by the workunit (FLOPS)

FPOPS(host): work cycles capacity of the host (FLOPS)

For each workunit result assigned to the host, we add the IWA to its' total work assigned.

$$TWA = \sum_{k=0}^{n} IWA(k)$$

Where:

TWA: total work assignment

IWA(k): incremental work assignment at turn k, k = {0, 1, 2… n}

n: number of workunit results assigned to a host

This value is checked before each result is assigned to a host, as a way to ensure that a host isn't given more work than it requested. For computers in a dedicated environment, the amount of work requested should generally correspond to the computer's full capacity, at a certain point in time, determined by the client application.

The second factor, *fairness*, concerns the fairness of work distribution, given the new queue based scheduling framework developed. In the native scheduler, work is distributed on first fit, first served basis. Any host requesting work that matches the requirements for a specific jobs, gets the task. While using a queue, this is not practical. The scheduler runs at predefined regular time intervals, or based on a threshold factor that limits the number of hosts of on the waiting queue before the scheduler runs. In either case, whenever the scheduler runs, it generally has several

hosts on a list waiting for work. To assign work fairly to these hosts in a manner that ensures overall project progression, a round robin scheme was used, where workunit results were interleaved between hosts on the waiting queue that were capable of taking them. The pseudo code for the scheduling scheme is as follows:

```
foreach wuResult in wuResults:
   assigned = False

   foreach request in requests:

   if (request.currentAssignment + wu.workEstimate) <=
request.workRequested:

      assign work to host (request)
      request.currentAssignment += wu.workEstimate
      assigned = True
      break

   if assigned == True:
       //push next host to the front – round robin
       move host that was assigned work to the bottom of the list
```

Figure 10 Round Robin Scheduling Pseudo Code

The third factor assessed by the scheduler is the hosts' *reliability*. Reliability here is a measure of the host's predicted ability to successfully complete the tasks that it is assigned by the scheduler. In the scheduler devised, reliability is estimated based on a single factor: hosts' recent availability. As stated in section 8.2, the logging frequency of the thermal server is once every 60 seconds. The thermal server determines the hosts' availability status by using the configured threshold temperature, and logs this information to 2 different files: one with for historical records, and a second for recent records only. With the frequency set to once every minute, and maxTAGRecords flag set to 30, all hosts collect the thermal data pertaining to the past half an hour before sending a request to the BOINC server. The reason why only recent thermal information was used is that, despite providing insight into trends, historical data does not provide as highly accurate estimates of the hosts' immediate future availability with low standard deviations as recent data does. This is supported by forecasting principles, which dictate that relevant data in forecasting analysis is generally recent data, as opposed to historical data, which is better suited for trend analysis [21][22].

The formula for reliability is given below:

$$Reliability = \frac{Number\ of\ minutes\ available}{Number\ of\ minutes\ measured}\ x\ 100$$

The scheduler has a threshold parameter to assess reliability. This parameter was set to 66.7%. Therefore, only hosts that had been available for at least 20 minutes out of the past 30 minutes were deemed reliable enough to receive new jobs. Conceptually, this factor should assist in predicting whether or not the host will be in a viable environment for the coming time. Had the host been under a non-viable period for the past 10 out of 30 minutes, it was still deemed viable. However, if it remained in such condition progressively over time, the server would become aware of the fact, as the host would check in; and the scheduler would not assign it work, because of the hosts' low availability score. The developed scheduler was used in the test environment, to assign results/jobs generated by the UppercaseWG engine to the 3 hosts in the setup.

8.6 Dashboard

The dashboard was one of final stages of development in this project. Its goal was to allow people to observe the behavior of the system over time, within the context of: temperature logging, hosts' reliability, and work assignment. The dashboard was built as a web based application. It was designed to regularly read the temperature, availability, and work assignment logs from the thermal server and the scheduling application, compute statistical measures for the day, and display this information in an intuitive and meaningful manner to users. Figure 11 shows a snapshot of the dashboard, taken March 8, at 2.22PM.

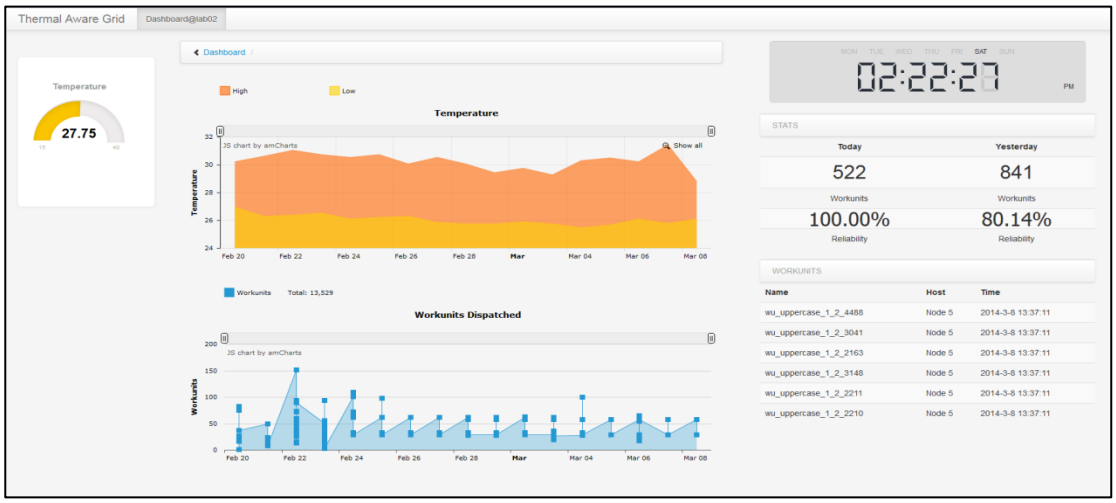


Figure 11 TAG Dashboard for HPC lab 2 (snapshot)

A snapshot of the daily statistics analyzed by the dashboard is shown in Figure 12. The table displayed compares the total number of workunits and overall lab reliability, measured as an average of all 3 nodes, for that day and the previous one.

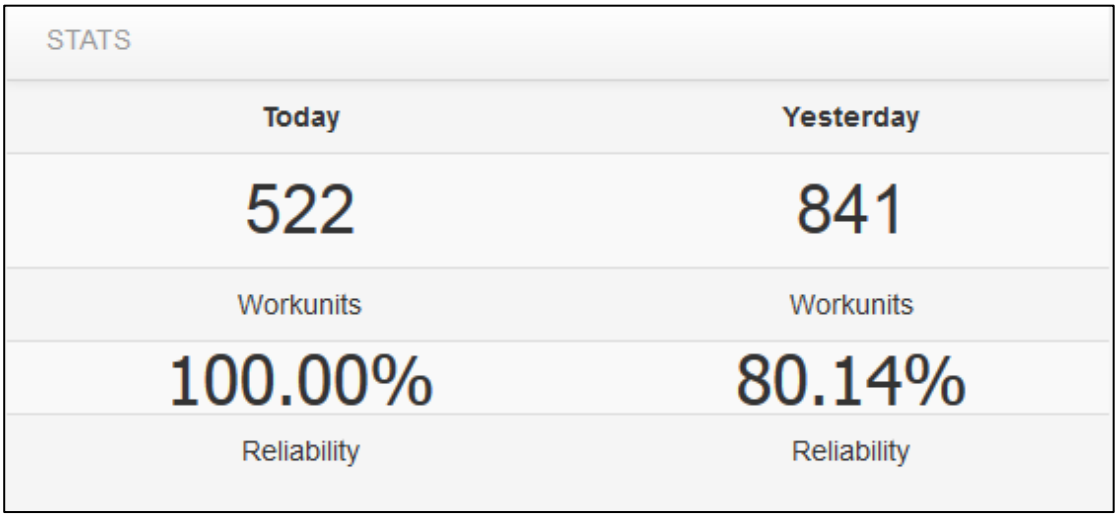


Figure 12 Dashboard stats display

Figure 13 shows the temperature log displayed on the dashboard. The chart shows daily minimum and maximum temperatures for the lab, with a highlight on the values for March 1st, 2014.

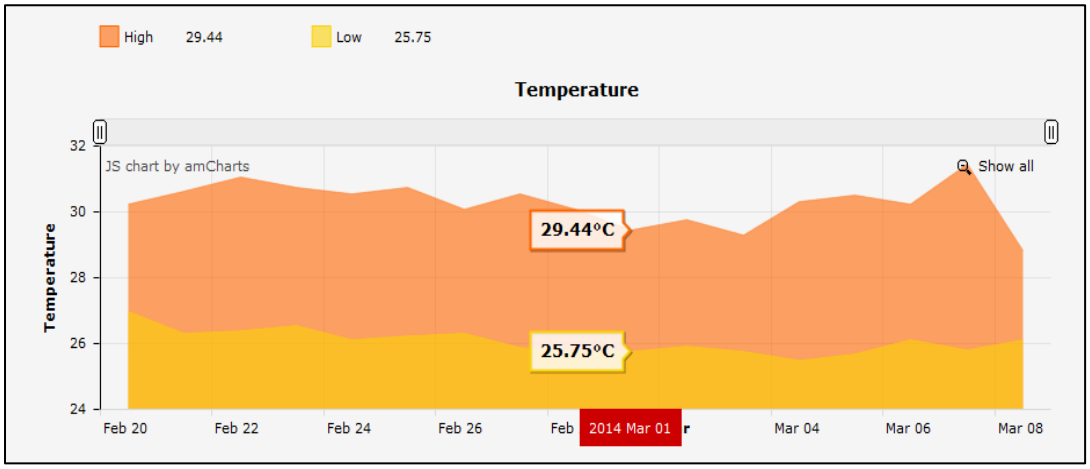


Figure 13 Dashboard temperature chart display: daily minimum and maximum

Figure 14 shows similar chart: the workunits dispatched dispalyed on the dashboard. It maps the hourly count of workunits dispatched to all nodes for each day. The snaphost shows the highest value of workunits sent at the peak hour on March 2nd.

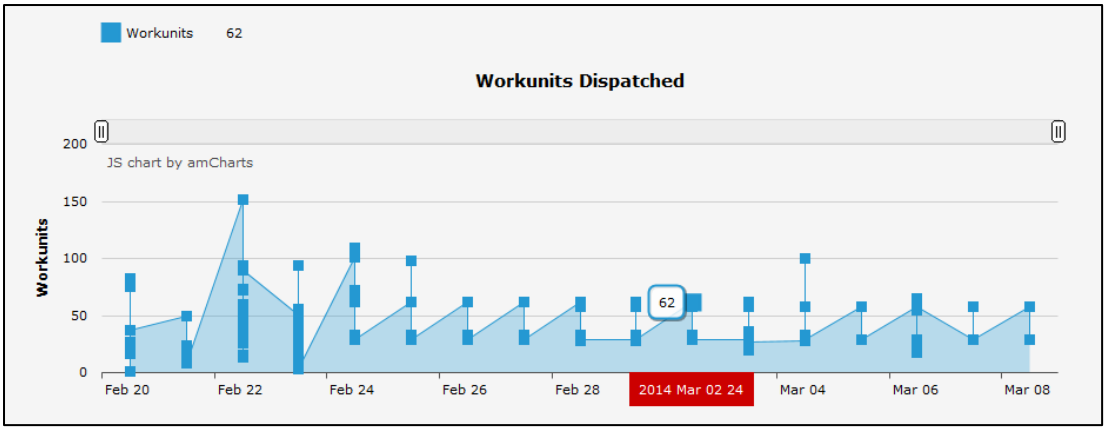


Figure 14 Dashboard workunits dispatched: total units dispatched per hour on different days

With the dashboard display, administrators can have a live feed to the grid's current and past performance. The usage of charts illustrates the behavioral pattern of the temperature, and work distribution over time. In addition, the zoom-in feature of the charts allows for users to zero in on a particular date and time, and observe the temperature readings and work distribution for that particular period alone.

Another piece of information available to users on the dashboard is the log of work assignment by the server, as in Figure 15.

WORKUNITS

Name	Host	Time
wu_uppercase_1_2_11763	Node 5	2014-4-6 0:59:7
wu_uppercase_1_2_11762	Node 6	2014-4-6 0:52:57
wu_uppercase_1_2_11761	Node 7	2014-4-6 0:48:57
wu_uppercase_1_2_11760	Node 5	2014-4-6 0:41:30
wu_uppercase_1_2_11759	Node 7	2014-4-6 0:36:40
wu_uppercase_1_2_11758	Node 5	2014-4-6 0:29:15

Figure 15 Dashboard workunits assignment log

8.7 Thermal Scheduling: Observations

The thermal scheduling application designed for the testing environment makes use of the concept of reliability presented in section 8.5. The time frame of the data that each host provides to the server corresponds to the availability logs of the past half hour (30 minutes), from the moment each request is made. The testing deployment began operating on the 20th of February 2014. Since then, the system has been logging both daily temperatures, and workunits assignments. Figure 15 shows a graph of total workunits dispatched during a 15 day period, from March 1 to March 15; and Figure 16 shows temperature logs for the same time period.

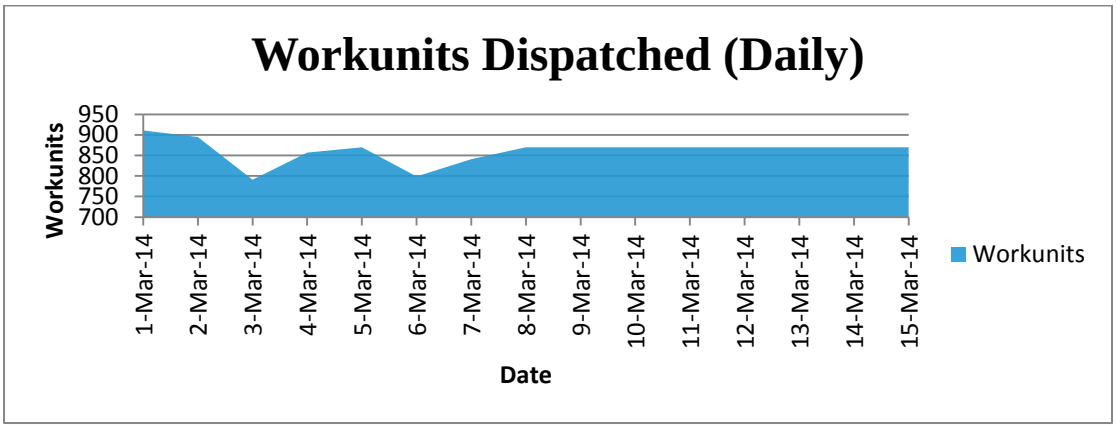


Figure 16 Workunits Dispatched, March 1 - March 15

The average number of workunits distribute to the 3 nodes in the environment is about 800, which gives us a work distribution of about 266 and 267 units per host.

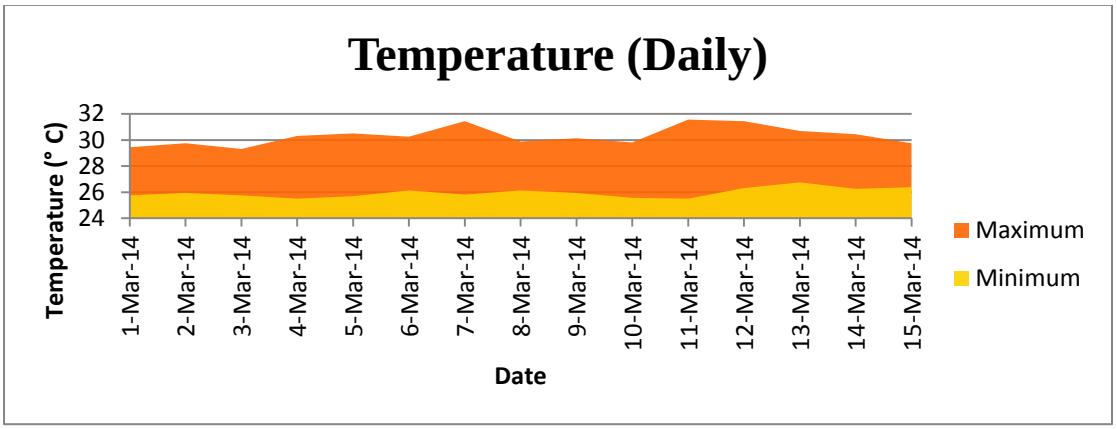


Figure 17 Daily Temperature, March 1 - March 15

The logs in Figure 17 show the minimum and maximum temperatures registered throughout the day. They do not, however, indicate for how long each temperature period lasted, and therefore cannot tell us how long each viable period lasted. The day with a highest registered temperature could also be the day with the lowest registered temperature. What needs to be assessed, therefore, is the relationship between the periods during which the temperature remained under the threshold, as well as the periods where it went over, and work distribution, as a factor of reliability.

To derive this information, we must analyze measured reliability against the workunits dispatched, and temperature logs shown in Figures 18 and 19, respectively. We shall look at hourly logs for fine detailed analysis.

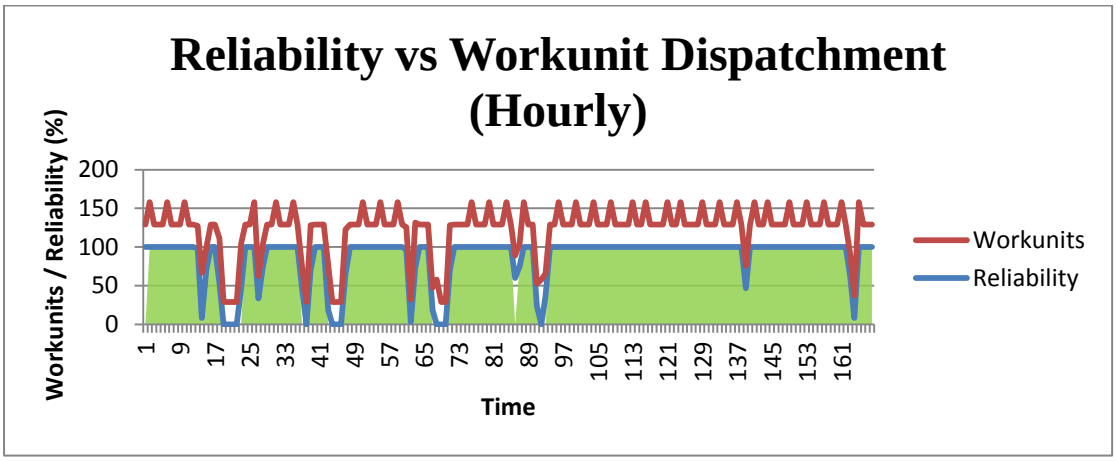


Figure 18 Reliability vs Workunit dispatchment, March 12 - March 18

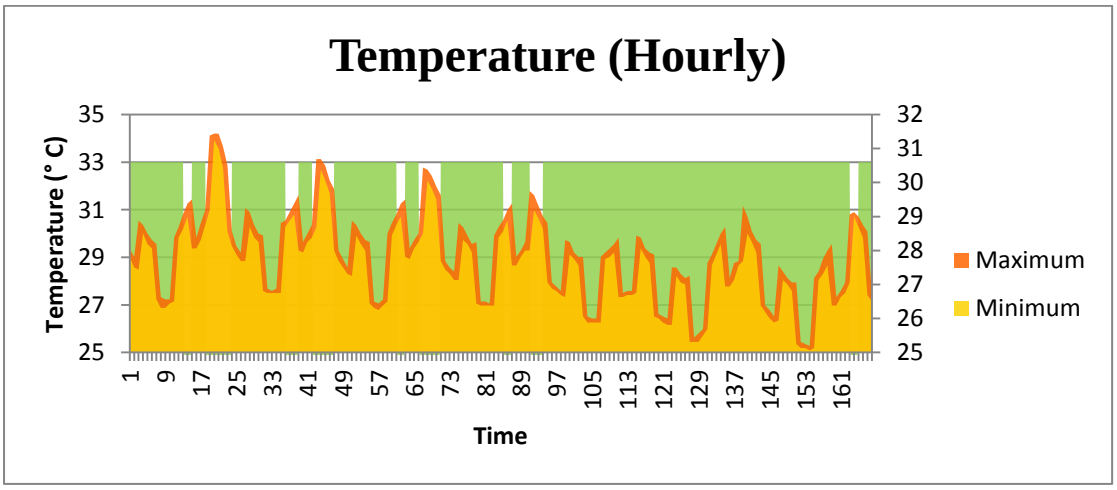


Figure 19 Hourly temperature, March 12 - March 18

The green area under the charts represents time periods when average room temperature for the hour was below the threshold of 29° Celsius, and the machines were thus considered to be available. Visual observation indicates that the number of workunits dispatched and workunit distribution seem to move in tune with each other. As reliability remains stable, so does the number of workunits dispatched. When reliability drops, the number of workunits dispatched also drops, by the exact same degree. If we analyze the temperature logs in Figure 19 and compare them against the reliability in Figure 18, we can observe that reliability is directly correlated to the temperature. The hours with the highest registered temperature have the lowest reliability grades, including 0%, and the hours with the lowest temperature have the highest reliability grades, peaking at 100%.

As indicated earlier, so as long as a machine is deemed unavailable, it does not receive any jobs. After being deemed unreliable, when a machine enters a viable temperature period, its reliability gradually increases by the minute, and when it reaches the threshold condition of 66.7%, it starts receiving work once again, in a non-linear fashion, i.e. the number of workunits dispatched to it does not depend on the node's actual reliability score. However, machines with higher reliability are placed on top of the work distribution list, and are thus are the first ones to receive work.

8.8 Mass Deployment

With the observations that work can be intelligently assigned on a grid with the modified version of BOINC, and developed thermal scheduler, the next issue of concern would be mass deployment of client applications to the lab computers on campus.

There are several tools available for mass software deployment, used by system administrators and managers. Some of these tools, like Active Directory, PSExec, and AutoIt are free, while others are not. Ideally, mass deployment should consist of a 2 stage process: software repackaging and deployment.

In first stage, software repackaging, the application's setup process is re-redesigned so as to not require any user input, thus removing the need for the physical presence of

the administrator during installation. Some setup applications require the user to specify the installation directory, select the packages to be installed, key-in license keys, etc. These details have to be pre-configured, so that setup wizard/process will not ask the user for them.

The second stage, deployment, is where the application is actually distributed to the target computers and remotely installed. If an application has already been pre-packaged for a "silent" installation, mass deployment would boil down to copying the setup packages to targeted remote machines, and activating them.

The processes involved in both stages are highly dependent on the actual setup program of the target application, and the environment of the target machines, i.e. operating system, user privileges, etc. On Windows, applications packaged with the native installation wizard framework can easily be configure for silent installs, provided that their distributors incorporate the necessary configuration parameters. Not all setup wizards, however, provide mechanisms for silent configuration. This, unfortunately, is the case of BOINC. There are, nonetheless, alternatives to circumventing this shortfall. AutoIt, for example, provides scripting facilities to emulate user input on a remote machine, when installs are not possible to configure at the setup level. The system allows users to pre-define answers for predetermined action steps in the installation process. Below is a sample script that can be used to install an early client version of BOINC on a remote machine.

AutoIt BOINC Installation Script

```
#RequireAdmin

Run("boinc_6.12.34_windows_intelx86.exe")

WinWaitActive("BOINC - InstallShield Wizard", "Welcome to
the InstallShield Wizard for BOINC")

Send("!n")

WinWaitActive("BOINC - InstallShield Wizard", "License
Agreement")

Send("!a")

Send("!n")

WinWaitActive("BOINC - InstallShield Wizard", "These are
the current installation options")
```

```
Send("{ENTER}")

ControlClick("BOINC - InstallShield Wizard", "Allow all
users on this computer to control BOINC", 2598)

WinWaitActive("BOINC - InstallShield Wizard", "Customize
installation options")

Send("!n")

WinWaitActive("BOINC - InstallShield Wizard", "Ready to
Install the Program")

Send("!i")

WinWaitActive("BOINC - InstallShield Wizard",
"InstallShield Wizard Completed")

Send("!f")

WinWaitActive("BOINC Installer Information", "You must
restart your system")

Send("!y")
```

The script defines actions, e.g. run “boinc_6.12.34_windows_intelx86.exe”, and indicates which screens or messages the system should wait for, as well as the response it should send or action it should perform once the awaited screen or message is displayed. Installation though, is not the only part of mass deployment. One also needs to configure the BOINC client machines to connect to a specific project. AutoIt can also be used to achieve this purpose. A similar script can define how the program should be configured from startup to connect to the target BOINC server.

```
Run("C:\Program Files\BOINC\boincmgr.exe")

WinWaitActive("BOINC Manager")

Send("!t")

Send("{ENTER}")

WinWaitActive("BOINC Manager", "Add project or account
manager")

Send("!n")

Send("{TAB}")
```

```
Send("[PROJECT's URL]")
Send("!n")
WinWaitActive("BOINC Manager", "&Yes, existing user")
Send("!y")
Send("{TAB}")
Send("[USERNAME]")
Send("{TAB}")
Send("[PASSWORD]")
Send("!n")
WinWaitActive("BOINC Manager", "Project added")
Send("!f")
WinClose("BOINC Manager")
```

With the minor requirements of copying the installation files to the lab computers via network shared folders and updating one of the client configuration files, i.e. cc_config.xml, the 2 scripts above greatly minimize the effort required for deploying client applications onto the campus computer labs.

CHAPTER 5
CONCLUSION AND RECOMMENDATAIONS

A grid deployment making use of idle periods of lab computers can make it possible for one to leverage vast amount of existing resources, at a low cost; especially for an academic institution. However, one needs to concern themselves first, with the viability of using such machines, which are placed in non-dedicated environments for a greater portion of time than not.

Green algorithms do not just help minimize energy consumption, but in certain contexts, they can help preserve computers by safeguarding them against highly potential hazards, such as overheat. If embedded into stable grid frameworks, they make it possible to take advantage of freely available resources with less concern over hardware damage. Further, making this integration dynamic/pluggable allows for many to use the same grid framework to address different environmental concerns.

One of the primary goals of this project, i.e. the enablement of dynamic scheduling at runtime within BOINC, was completed in the first stage, tested, and observed. Both client and server machines were modified to support and manage thermal data retrieved from a thermal server. The BOINC server also accurately fed this information, along with other details of the hosts and workunit results available to a developed external scheduling application. The second major goal of the project, which was to have a pilot project of the system deployed, was also achieved. With the pilot project running, performance of a thermal aware grid distribution system was observed and analyzed. Visual evaluation of data showed that the system could intelligently assign workload based on the thermal conditions of lab computers.

A dashboard was developed as a monitoring tool to accurately relay vital information from the thermal server and scheduler in real-time. Although the system was not

deployed one of the academic laboratories, for organizational reasons, ease of mass deployment was also studied, and conclusively determined to be fairly high.

Despite both primary goals of the system having been reached, there is much room for further enhancement. For instance, the application used throughout testing was overly simplistic. A more accurate study should consider not only different types of applications with greater runtimes and requirements, but also with a fair share of both CPU and I/O intensive cycles. Similarly, all hosts used in the testing environment were homogenous, and located in the same environment. It would be interesting to observe the behavior of the thermal scheduler for a deployment with computers located in at least 2 distinct thermal environments, and of different architectures as well as platforms.

Another aspect that could be given priority in future work is the use of more refined thermal-aware scheduling algorithms. Although the basic scheduler developed here does intelligently assign work load based on hosts' thermal availability, its efficiency has not been compared nor benchmark against that of any existing thermal scheduler, seen as that the prime directives of the project were to evaluate the system's extensibility in alternating between scheduling applications at run time, and supporting sensor based thermal scheduling frameworks only.

Finally, to observe actual performance on a larger-scale, the test deployment should be expanded to include computers in laboratories that are used for academic purposes during the day. Monitoring and evaluating the thermal availability, work distribution and reliability of machines in such environment would give greater insight into the development of more fine-tuned work scheduling algorithms the grid's context. It is the hope of the author that the work done so far can provide sustainable grounds for the enhancements prescribed.

REFERENCES

[1] Rodrigues, R., and Druschel, P. (October, 2010). Peer-to-Peer Systems. Communications of the ACM. 53 (10)

[2] Beberg, A.L., Ensign, D.L., Jayachandran, G., Khaliq, S., Pande, V. S. (2009). Folding@home: Lessons From Eight Years of Volunteer Distributed Computing. IEEE International Symposium on Parallel & Distributed Processing, 2009. IPDPS 2009.

[3] Giovannozzi, M., Harutyunyan, A., Hoimyr, N., Jones, P. L., Kerneyeu, A. , Marquina, M. A., McIntosh, E. , Segal, B., Skands, P. , Grey, F. , Lombrana Gonzalez, D., Rivkin, L. , Zacharov, I.(2012). LHC@Home: A Volunteer Computing System for Massive Numerical Simulations of Beam Dynamics and High Energy Physics Events. International Particle Accelerator Conference (IPAC '12).

[4] Zhu, T., Shu, C., & Yu, H. (2011). Green Scheduling: A Scheduling Policy for Improving the Energy Efficiency of Fair Scheduler. 12th International Conference on Parallel and Distributed Computing, Applications and Technologies. pp319-326, 2011

[5] Zhang, L. M., Zhang, Y., Li, K. (2010). Green Task Scheduling Algorithms with Speeds Optimization on Heterogeneous Cloud Servers. 2010 IEEE/ACM International Conference on Green Computing and Communications & 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing

[6] Anderson, D.P., Christensen, C., & Allen, B. (2006). Designing a Runtime System for Volunteer Computing

[7] Berman, F., Fox, G., & Hey, T. (2002). The Grid: past, present, future. Grid Computing – Making the Global Infrastructure a Reality

[8] Okitsu, J., Naono, K., Sulaiman, S. A, Zakaria, N., Oxley, A. (2012). Towards Greening a Campus Grid: Free Cooling During Unsociable Hours. Unpublished

[9] Anderson, D. P., Korpela, E. & Walton, R. (n.d.). High-Performance Task Distribution for Volunteer Computing. Space Sciences Laboratory. University of California, Berkeley, unpublished

[10] Kosar, T. & Balman, M. (2008). A new paradigm: Data-aware scheduling in grid computing. Future Generation Computer Systems. 25(2009). pp406-413

[11] Goiri, I., Haque, M. E., Le, K., Beauchea, R., Nguyen, T. D., Guitart, J., Torres, J., Bianchini, R. (2011). GreenSlot: Scheduling Energy Consumption in Green Datacenters. SC '11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis

[12] Liu, S., Qiu, M. (2010). Thermal-Aware Scheduling for Peak Temperature Reduction with Stochastic Workloads. Work-in-Progress Proceedings

[13] Fisher, N., Chen, J., Wang, H., and Thiele, L. (2009). Thermal-Aware Global Real-Time Scheduling on Multicore Systems. 15th IEEE Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009.

[14] Shi, B., and Srivastava, A. (2010). Thermal and Power-Aware Task Scheduling for Hadoop Based Storage Centric Datacenters. Green Computing Conference, 2010 International

[15] Mukherjee, K., Khullerm, S., Deshpande, A.(2012). Saving on Cooling: The Thermal Scheduling Problem. SIGMETRICS '12 Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems. pp397-398

[16] Mukherjee, K., Khuller, S., Deshpande, A. (2013). Algorithms for the Thermal Scheduling Problem. 2013 IEEE 27th International Symposium on Parallel and Distributed Processing

[17] Anderson, D. P. (n.d.). BOINC: A System for Public-Resource Computing and Storage. Space Sciences Laboratory. University of California at Berkeley

[18] Dlaverty, (2009). Recommended Server Room Temperature. Openxtra. Retrieved December 23, 2013 from http://www.openxtra.co.uk/articles/recommended-server-room-temperature

[19] Miller, R. (2008). Google: Raise Your Data Center Temperature. Data Center Knowledge. Retrieved December 23, 2013, from

http://www.datacenterknowledge.com/archives/2008/10/14/google-raise-your-data-center-temperature/

[20] ITWatchDogs. (2013). IT Watch Dogs. Retrieved December 23, 2013, from http://www.itwatchdogs.com/computer-room-temperature

[21] Armstrong, J.S. (2001). Principles of Forecasting: A Handbook for Researchers and Practitioners. Springer Science+Business Media

[22] Tay, F.E.H. & Cao, L.J (2002). Modified Support Vector Machines in Financial Time Series Forecasting. Neurocomputing. 48. pp847–861

APPENDICES

APPENDIX 1 - WORK BREAKDOWN STRUCTURE

Task #	*Task Name*	*Duration*	*Start Date*	*Finish Date*
1	Initiating	14	9/25/2013 8:00	10/14/2013 16:00
2	Identify project title	10	9/25/2013 8:00	10/8/2013 16:00
3	Submit project title	0	10/9/2013 8:00	
4	Planning	56	9/25/2013 8:00	12/11/2013 16:00
5	Develop project extended proposal	26	9/25/2013 8:00	10/30/2013 16:00
6	Analyze requirements	7	10/30/2013 8:00	11/7/2013 16:00
7	Specify system modification requirements	3	11/7/2013 8:00	11/11/2013 16:00
8	Define scope	7	11/11/2013 8:00	11/19/2013 16:00
9	Verify scope	7	11/11/2013 8:00	11/19/2013 16:00
10	Develop a schedule	1	10/28/2013 8:00	10/28/2013 16:00
11	Executing	188	9/25/2013 8:00	6/13/2014 16:00
12	Feasibility study	7	11/19/2013 8:00	11/27/2013 16:00
13	System Architecture	14	11/27/2013 8:00	12/16/2013 16:00
14	Define interfacing requirements	4	12/12/2013 8:00	12/17/2013 16:00
15	System design	7	12/10/2013 8:00	12/18/2013 16:00
16	Validate solution	1	12/19/2013 8:00	12/19/2013 16:00
17	Present solution and feasibility to SV	1	12/20/2013 8:00	12/20/2013 16:00

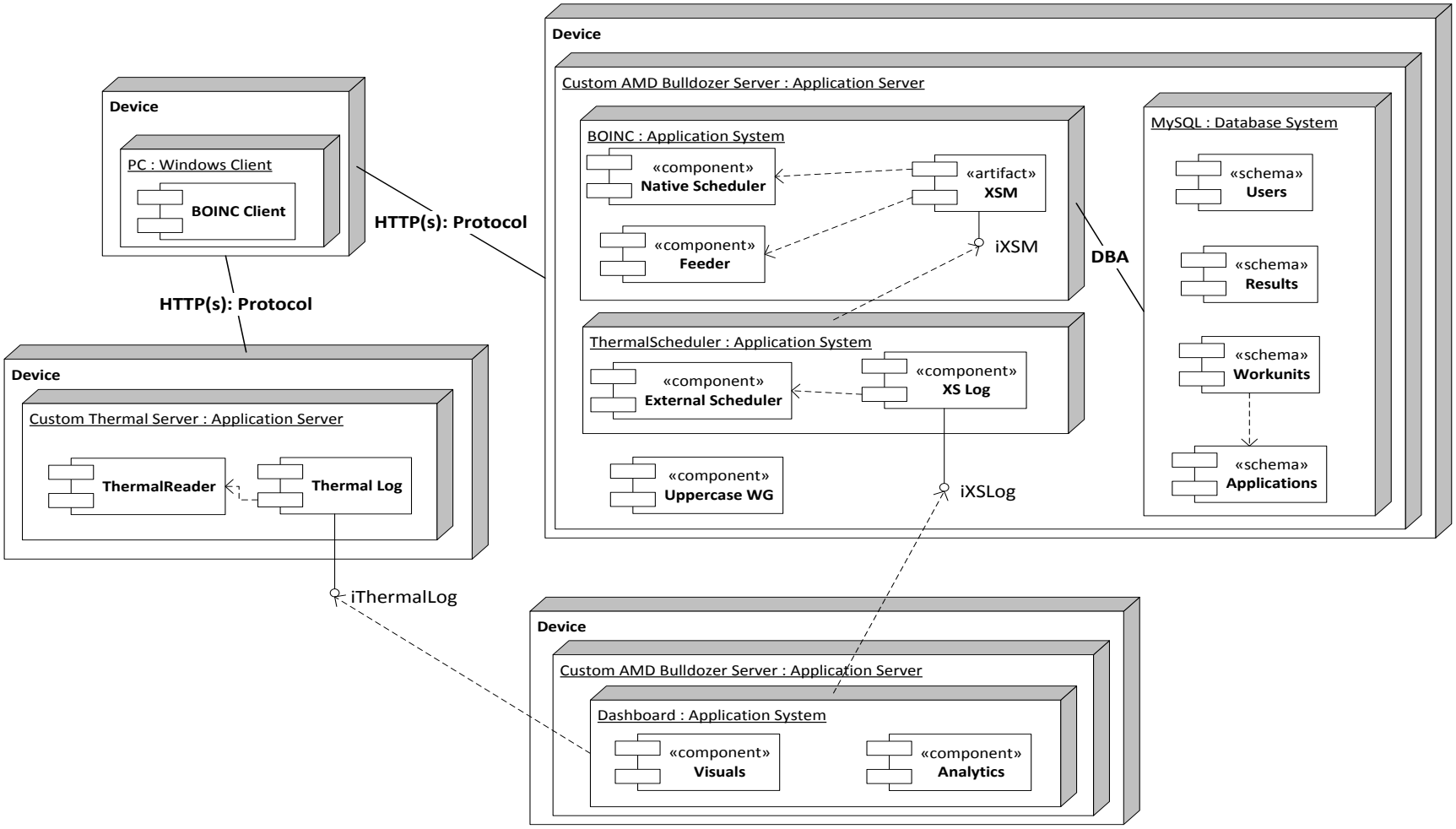
18	Present system conceptual design	1	12/20/2013 8:00	12/20/2013 16:00
19	Create quality metrics	1	9/25/2013 8:00	9/25/2013 16:00
20	Build/Modify modules	45	1/8/2014 8:00	3/11/2014 16:00
21	Perform unit testing	45	1/8/2014 8:00	3/11/2014 16:00
22	Testing & Evaluation	24	2/18/2014 8:00	3/21/2014 16:00
23	Integration tests of relevant modules	7	2/18/2014 8:00	2/26/2014 16:00
24	System testing	4	2/28/2014 8:00	3/5/2014 16:00
25	Application effectiveness testing	3	3/10/2014 8:00	3/12/2014 16:00
26	Measure system impact on processes	3	3/11/2014 8:00	3/13/2014 16:00
27	Define implementation strategy	5	3/18/2014 8:00	3/24/2014 16:00
28	Closing	25	4/1/2014 8:00	5/5/2014 16:00
29	Close project	9	4/15/2014 8:00	4/25/2014 16:00
30	Create report	19	4/1/2014 8:00	4/6/2014 16:00
31	Submit extended proposal	0	10/30/2013 8:00	
32	Submit interim report	0	12/30/2013 8:00	
33	Submit progress report	0	2/10/2014 8:00	
34	Submit technical report	0	4/14/2014 8:00	
35	Viva presentation	0	4/28/2014 8:00	
36	Submit project dissertation	0	4/28/2014 8:00	

APPENDIX 2 – TEMPLATE XML FILES

External Scheduler (XS) Output

```
<xs_output>
  <assignment>
    <host_id>4</host_id>
    <wu_id>4</wu_id>
    <result_id>25</result_id>
  </assignment>
  <assignment>
    <host_id>3</host_id>
    <wu_id>4</wu_id>
    <result_id>26</result_id>
  </assignment>
  <assignment>
    <host_id>5</host_id>
    <wu_id>5</wu_id>
    <result_id>27</result_id>
  </assignment>
  <assignment>
    <host_id>4</host_id>
    <wu_id>5</wu_id>
    <result_id>28</result_id>
  </assignment>
  <assignment>
    <host_id>3</host_id>
    <wu_id>5</wu_id>
    <result_id>29</result_id>
  </assignment>
  <assignment>
    <host_id>5</host_id>
    <wu_id>5</wu_id>
    <result_id>30</result_id>
  </assignment>
  <assignment>
    <host_id>4</host_id>
    <wu_id>5</wu_id>
    <result_id>31</result_id>
  </assignment>
  <assignment>
    <host_id>3</host_id>
    <wu_id>5</wu_id>
    <result_id>32</result_id>
  </assignment>
</xs_output>
```

APPENDIX 3 – THERMAL AWARE GRID SYSTEM DEPLOYMENT DIAGRAM



APPENDIX 4 – SAMPLE LOGS

Temperature Log

Day Month Year Hour Minute Temperature

12 3 2014 0 0 28.06

12 3 2014 0 1 28.06

12 3 2014 0 2 28.06

12 3 2014 0 3 28.0

12 3 2014 0 4 28.0

12 3 2014 0 5 28.0

12 3 2014 0 6 28.0

12 3 2014 0 7 28.0

12 3 2014 0 8 28.0

12 3 2014 0 9 28.0

12 3 2014 0 10 28.06

Availability Log

Day Month Year Hour Minute Availability

12 3 2014 0 0 1

12 3 2014 0 1 1

12 3 2014 0 2 1

12 3 2014 0 3 1

12 3 2014 0 4 1

12 3 2014 0 5 1

12 3 2014 0 6 1

12 3 2014 0 7 1

12 3 2014 0 8 1

12 3 2014 0 9 1

12 3 2014 0 10 1

Assignments Log

HostId WorkunitId WorkunitName ResultId Year Month Day Hour Minute Second

5 5121 wu_uppercase_1_2_5120 9453 2014 3 12 0 16 22

5 5122 wu_uppercase_1_2_5121 9454 2014 3 12 0 16 22

5 3869 wu_uppercase_1_2_3868 9456 2014 3 12 0 16 22

5 3873 wu_uppercase_1_2_3872 9457 2014 3 12 0 16 22

5 3875 wu_uppercase_1_2_3874 9458 2014 3 12 0 16 22

5 4419 wu_uppercase_1_2_4418 9459 2014 3 12 0 16 22

5 5123 wu_uppercase_1_2_5122 9460 2014 3 12 0 16 22

5 5124 wu_uppercase_1_2_5123 9461 2014 3 12 0 16 22

5 5125 wu_uppercase_1_2_5124 9462 2014 3 12 0 16 22

5 4096 wu_uppercase_1_2_4095 7793 2014 3 12 0 16 22

5 4097 wu_uppercase_1_2_4096 7794 2014 3 12 0 16 22

APPENDIX 5 – SYSTEM TESTING: TEST CASES

Table 1 – System Test Case 1

Test Case Identifier	ST1
Test Module	XSM (Server)
Purpose	Verify that the unit can successfully dump hosts' request and available jobs' data to the xs_input.xml file
Input	Waiting queue, and Jobs list (on shared memory)
Expected Output	Valid XML file is created, named xs_input.xml, with valid data of jobs available and data of hosts that have sent work requesting
Intercase Dependencies	-

Table 2 – System Test Case 2

Test Case Identifier	ST2
Test Module	XSM (Server)
Purpose	Verify that the unit properly calls the user defined external scheduling application, and waits for it to return control
Input	A valid XS on the xscheduler folder, and its full path specified in the config.xml file under the, a host request
Expected Output	Test XS program runs, and generates expected xs_output.xml file for the XSM
Intercase Dependencies	ST1

Table 3 – System Test Case 3

Test Case Identifier	ST3
Test Module	XSM (Server)
Purpose	Verify that the unit detects when the specified XS program is not present, and calls the native scheduling sequence
Input	Invalid XS path specified in the config.xml file, a host request
Expected Output	Native BOINC scheduler sequence is initiated
Intercase Dependencies	-

Table 4 – System Test Case 4

Test Case Identifier	ST4
Test Module	XSM (Server)
Purpose	Verify that the unit reads the XS work assignments in the xs_output.xml, and assigns jobs to hosts as specified by the XS
Input	A valid xs_output.xml, with valid job assignments (i.e. assignment of available jobs) to hosts on the waiting queue
Expected Output	XSM assigns jobs to hosts, and hosts receive jobs the next time they send a work request
Intercase Dependencies	ST2

Table 5 – System Test Case 5

Test Case Identifier	ST5
Test Module	XSM (Server)
Purpose	Verify that the unit logs the timestamp after calling the XS application
Input	A valid XS speficied in the config.xml file, and a host request
Expected Output	A file named xs_last_run is created/update with a valid timestamp
Intercase Dependencies	-

Table 6 – System Test Case 6

Test Case Identifier	ST6
Test Module	XSM (Server)
Purpose	Verify that the unit properly parses hosts' availability data from their request files
Input	A valid host request, with thermal availability data
Expected Output	Thermal data is parsed, and logged
Intercase Dependencies	-

Table 7 – System Test Case 7

Test Case Identifier	ST7
Test Module	Client
Purpose	Verify that the unit downloads thermal data from the thermal server
Input	A valid thermal profile URL
Expected Output	Host gets thermal data, and logs it
Intercase Dependencies	

Table 8 – System Test Case 8

Test Case Identifier	ST8
Test Module	Client
Purpose	Verify that the unit parses downloaded thermal data, and appends it to its work request
Input	A valid thermal profile URL
Expected Output	Hosts appends retrieved thermal data to its work request file, request.xml
Intercase Dependencies	ST7

Table 9 – System Test Case 9

Test Case Identifier	ST9
Test Module	Client
Purpose	Verify that the unit sends a work request file with thermal data appended to it
Input	A valid thermal profile URL, and connection with thermal-aware server
Expected Output	Host sends requests, and receives server response successfully
Intercase Dependencies	ST8

Table 10 – System Test Case 10

Test Case Identifier	ST10
Test Module	Client
Purpose	Verify that the unit receives jobs assigned to it by an XS application, upon sending a second work request
Input	Two work requests from host to server, interleaved by at least 5 seconds
Expected Output	Host receives jobs from server, assigned by an XS
Intercase Dependencies	ST4, ST9

Table 11 – System Test Case 11

Test Case Identifier	ST11
Test Module	Client, Scheduler (Server)
Purpose	Verify that the unit sends completed job results to the server
Input	-
Expected Output	Server receives and acknowledges completed jobs from a client
Intercase Dependencies	ST10

Table 12 – System Testing Results

Test Case	Count	Pass	Fail	Error
ST1	5	5	0	0
ST2	5	5	0	0
ST3	5	5	0	0
ST4	5	5	0	0
ST5	5	5	0	0
ST6	5	5	0	0
ST7	5	5	0	0
ST8	5	5	0	0
ST9	5	5	0	0
ST10	5	5	0	0
ST11	5	5	0	0