# Clowder: A Software for Managing Cluster Of High Performance Computers

by

© *Samson Ugwuodo*

A thesis submitted to the

School of Graduate Studies

in partial fulfilment of the

requirements for the degree of

Master of *Science*

*Scientific Computing* Program

Memorial University of Newfoundland

*A*pril 2017

St. John's Newfoundland

## Abstract

Clowder is a system designed to help researchers in the Faculty of Engineering and Science to manage and control clusters of test machines. Some of the features that were missing in this system is the ability to reserve a computer for a certain period of time, a function that allows users to add more computers and manage their properties , and also a user interface where users can have interactive access to the system. Therefore, there was a need to upgrade and complete this system by developing software that provides the missing functionality. A web interface addresses the issue of user accessibility. Having a dynamic database system provides the functionality of keeping record of user activities, storing computer information and managing all reservations. These design approaches are achieved using SQL models, HTML templates and the Go programming language. Instead of performing several command line prompt statements, which is the current method in the use of Clowder system, the software provides flexible user access to the cluster of computers, a dynamic system management in a single software system as a research tool in the Faculty of Engineering and Science.

## Acknowledgements

I will like to express my humble thanks to my supervisor Dr. Jonathan Anderson for his overwhelming support, guidance, professional advise as a professor and his contribution through out this project.

I want to express my sincere appreciation to the School of Graduate Studies, the Department of Scientific Computing and Computer Engineering for their academical and financial support.

I would also like to thank the Head of Department Dr Ronald Haynes and Gail Kenny for their subsequent advice and support through out my in class academic works and other wise.

Finally, I am grateful to my family and God almighty for the love and encouragement that I have been blessed with through out this journey.

# Contents

**Abstract** **ii**

**Acknowledgements** **iii**

**List of Tables** **vi**

**List of Figures** **vii**

**1 Introduction** **1**

**2 Background** **4**

**3 Design and Requirements** **6**

3.1 Requirements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3.2 Design . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3.2.1 User Interface . . . . . . . . . . . . . . . . . . . . . . . . . . 8

3.2.2 Server . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

3.2.3 Database . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

3.2.4 Data flow and protocols . . . . . . . . . . . . . . . . . . . . 11

**4 Implemetation** **12**

4.1 Review . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

4.2 Implementation tools . . . . . . . . . . . . . . . . . . . . . . . . . . 12

4.3 Program Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

4.4 List of Funtions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

4.5 Description of Functions . . . . . . . . . . . . . . . . . . . . . . . . 15

4.6 Get Funtions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

4.7 Problems Encounted . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

**5 Evaluations** **23**

5.1 Test cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

5.2 Evaluations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

**6 Conclusion** **29**

**Bibliography** **30**

# List of Tables

4.1 Program package . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

4.2 List of Funtions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

5.1 List of Reservations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

5.2 test 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

5.3 test 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

# List of Figures

1.1 Design . . . . . . . . . . 3

3.1 Design . . . . . . . . . . 8

3.2 Server Activity Description . . . . . . . . . . 10

3.3 Database design . . . . . . . . . . 11

5.1 Reservation function . . . . . . . . . . 25

5.2 Before update . . . . . . . . . . 26

5.3 Editing machine details . . . . . . . . . . 27

5.4 Updated machine properties . . . . . . . . . . 27

5.5 Reservation function . . . . . . . . . . 28

# 1 Introduction

The use of high performance computers for research in the Faculty of Engineering and Scientific has increased, because of increase in research tasks, such as testing new operating systems. As researchers demand robust computer architectures, these computer systems and their properties also need to be expanded to accommodate this large tasks. As we expand the systems access and usage becomes more complicated to manage. Therefore we are faced with the issue of system management, system accessibility, and storage. These issues for example, are the inability to provide proper record of each computer system and their usage, the inability to provide flexible users access to the computers, and also the problem of monitoring the number of computers that are reserved by users. As a result of these challenges it is highly necessary to develop software that solves these problems for Faculty of Engineering and Science.

Clowder is a system designed to provide Preboot Execution Environment (PXE) for testing new operating systems through a Network File System (NFS) sever on the high performance computers. Previous work has been started by researchers in Engineering department to complete these system for several years, but yet there was more improvement yet to be made in order to complete it, as it lacks some necessary modern functionality and features, such as database system, interactive user interface and restrictions. As an example, it takes extra effort and manual command line statements to access a computer in the cluster during research. Another reason for improvement is to provide flexible and dynamic user interface rather than current manual, error-prone (SSH). Therefore the current state of Clowder is not flexible and convenient enough. So the main goal of this project is to develop software that addresses the issue of user accessibility, system management and automatic control protocol. Addressing all these issues will improve the performance of Clowder in speed , access and robustness, and therefore making it a completed software tool.

We have accomplished this goal by using a design approach that provides a web interface, a databases system and automatic command protocols. The database system is designed to store and keep record of the computer systems and user activities. The web interface enable users to have access to the Clowder system from different locations simultaneously via a web page, and also provide user with the system inventory and other user activities. This software provide the ability to adding new computer system to the cluster, and to modify their properties individually. Also it provides the ability to make reservations: to allow users to reserve a computer for a certain period of time, and able to end reservations as well. A function to allow user search the inventory with some specifications according to their demands.

As this software serves as a tool to manage the cluster of computers and user activities, it is important to know that reservations made, computer details, network interface cards and disks are stored as data. So all the computer system installed in the cluster with their names, vendors, memory size, architecture, and micro architecture is stored in the database. And same is applicable to the disks, network interface cards and any other devices that could be part of the cluster. Data is represented as variables in their various data types in the database scheme. All these information stored in the database tables servers as input data for the program and out put for the inventory. This database scheme allows user to add or update new machines installed in the cluster, and therefore provide data record for the inventory on the web interface.

The web interface serves as a platform for users to interact with the system and overview other users activities via a web page using designated URL. This interface has also replaced the SSH command line prompts which was the previous user interface for Clowder system. This choice provides flexible access and control to the system, by allowing users to log on to the system and make request of the inventory at any time through the web server. The following figure show a genaral description of the Clowder system:

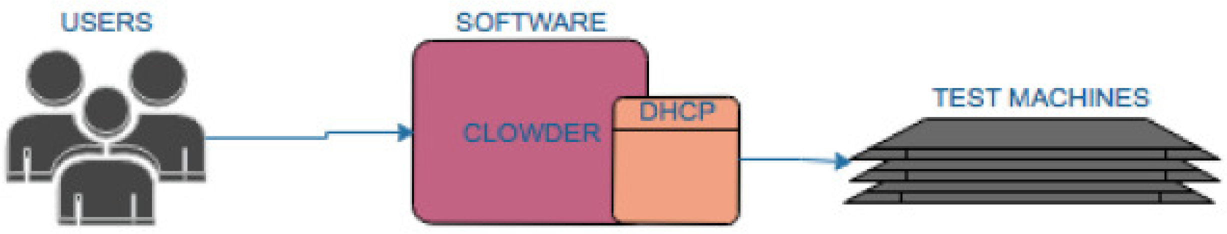


Figure 1.1: Design

The above figure showcases the background concept of the Clowder as a full system. However this concept has more detail as shown in Fig 3.1 that explain the software functionality and data flow between the user interface and database. Following this chapter, we explain in more detail about the background of this system and other component that made up the design.

# 2 Background

## Clowder

Clowder is made up of various components, and some of this components has been worked on by researchers in the department. The main goal of this project is to complete the Clowder system by developing the remaining components, which are to solve the issue of user accessibility and system management. Generally Clowder has been designed to manage access of cluster of test machines, mainly for testing new operating systems. But as this requires a flexible user interface and database to complete this components, we have designed a dynamic user interface and functioning database to accomplished this need. The background components of Clowder includes a DHCP server with Preboot Execution Environment (PXE) that allow the test machines to be boot up remotely by users for testing. The test machines do a PXE boot and by using the DHCP to get IP then request for files. Generaly a PXE is defined on a foundation of industry standard Internet protocols and services that are widely deployed using DHCP. This process forms an interactions between clients and servers. To ensure that the meaning of the client-server interaction is standardized as well, certain vendor option fields in DHCP protocol are used, which are allowed by the DHCP standard. For example, the Wireshark is a fully functional dissector in DHCP packet that detect bugs during this processes[1]. The operations of standard DHCP servers that serve up IP addresses will not be disrupted by the use of the extended protocol [6]. The client initiates the protocol by broadcasting a DHCPDISCOVER containing an extension that identifies the request as coming from a client that implements the PXE protocol. The client then discovers a Boot Server of the type selected and receives the name of an executable file on the chosen Boot Server [6]. These files are managed with a Network File System (NFS), which allow users to access files across networks. This processes serves as the main function of Clowder system as a testing tool, so the rest is the part which we have designed to complete this system as a full functional software tool for research. In the design of some front end functionality extended to back end, we used an SQL JOIN query for combination of different database data where our algorithms requires to merge multiple data for some query request. Generally a SQL join is a Structured Query Language (SQL) instruction to combine data from two sets of tables. There are different types of join query which includes INNER, OUTER, LEFT and RIGHT join.

# 3 Design and Requirements

## 3.1 Requirements

The objective of this software is to a dynamic user interface and a functional database system. With regard to that, the specification is made up of necessary features and functionality that appear on the user interface. The following are the major specification of this software:

- Data Inventory
- Searching Inventory(SI)
- Make Reservations
- Cancel Reservations
- Add Data
- Update Data

The user interface is able to list the inventory of data stored in the database in different categories. It is one of the features of the user interface because it presents the content of the database. Data inventory is required to retrieve information from the database and requested by users on the interface. For example, when users need to see the list of test machine in the system, this feature is repsonsible for providing the information already on the web interface. Necessary functionality are designed to help users access the inventory faster, one of them is the ability to sort the inventory cronologically or by data sequence. As there are lots of different data listed in the inventory, it is necessary to have a function that allow users to search information from the inventory. Searching the inventory helps the users to get specific data by typing in a set of query such dates, time, NFS root , PXE and memory sizes of a particular machine or reseravtions. This feature has a text feild for inserting text and search button for submitting request on the user interface. Users can search for a particular reservation by specifying a range of data they want to see, and this will present the lists of reservations made within that date and so on. Another example of searching inventory is where users can avalaible machines, this allows the user at all time to know which machine is free for reservation. Another requirement of this software is the ability to make reservations: users are able to choose a machine from the inventory and reserve it for a fixed date and time. This requirement on the high level is meant to allow user to a specific machine for running a particular task on that machine without interruption. When a reservation made is expired, the machine that was reserved becomes free for other users. As new machines are added to the cluster, the Add-data functionality allows users to add details of those machines to the database and update the inventory. Likewise, when the user want to change a certain properties of the machines such as, memory size or disk, the Update-data functionality helps to perform this action.

## 3.2 Design

The main idea of the software is to solve the issue of user interface, and system management. So we have designed this software with the approach to provide functional solution to the specifications. The design structure comprises the various segments and element of the user interface (front end) combined with the database (back end) and the control protocols. Below is a figure that describe the main structure of the design in terms of operation:

We designed a web interface that provides flexible and dynamic access to the system with interactive functionality. This web page contains inputs and output features. The reason for choosing a web page instead of other options is to provides online dynamic and simultaneous access to the system rather than performing manual command prompts (example: checking the usage of machine, properties of machine, and status of machine on Unix interface). Another reason is to have a functional real time user interface that present the activities of the Clowder. The web server is responsible for serving the web page all the resources requested by the users, and it serves

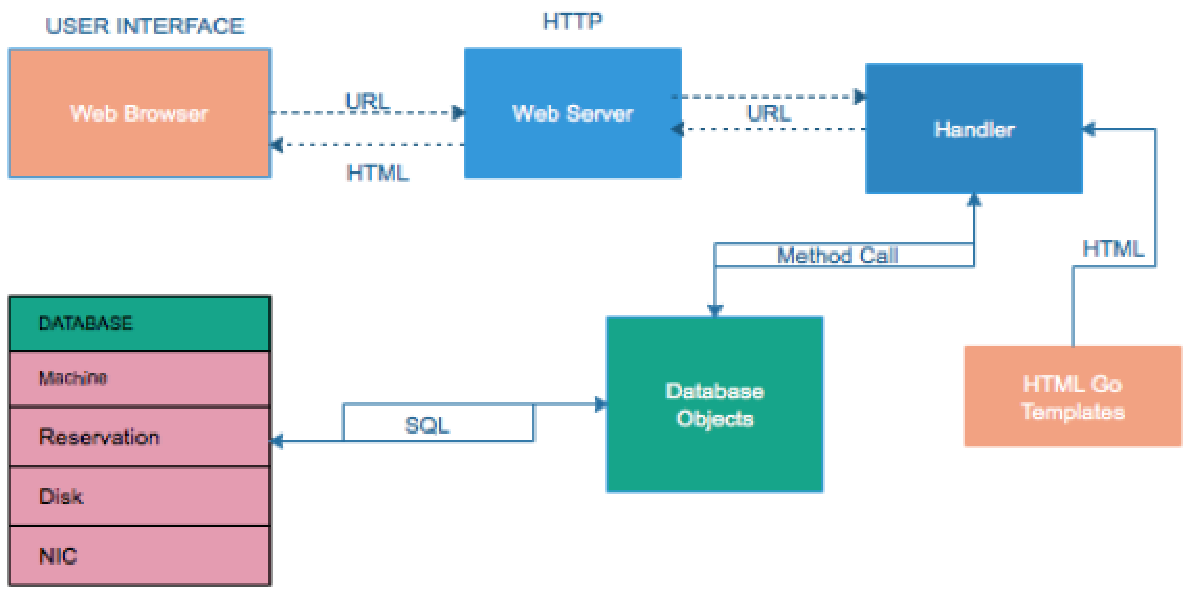


Figure 3.1: Design

as a channel between the user and the database. We designed an operational (event recording and system referencing) database that address the issue of system management. The database stores data from the web page and also retrieves data on the web page via the server. As the user log on to the their account on the web page the server will pass this information to the database which is controlled by the main program (command protocol). The command protocols is responsible for handling the user interface activities and executing the database queries.

### 3.2.1 User Interface

The web interface represents the user interface (UI) where users get access to the system. It is part of the design that represents the front end of the software.The web interface elements controls the basic functionality for data input and out put. It represent each feature as mentioned in the specification. This web interface structure is designed using HTML template as the front end and Go programming language as the back end. The web interface contains all the necessary elements required to have dynamic and flexible access to the system. These elements includes:

- Forms
- Input controls

#### Forms

Forms are one the elements that provides the ability to input data to the database.The data is submitted through a HTML post request [7]. The post forms are used for submitting data such as machines and reservations details. The form is designed with elements which includes; id and name attributes for identifying data, text fields for collecting data, select menus for selecting data options and submit button for submitting the data. When data is submitted with HTML form, the Go HTTP framework in the web server call the Go function handler to process and parse this data to the database. After the data (example: a database query) is processed, the Go program generates an HTML page (such as machines inventory), which the server returns to the web interface. In this system design, the forms are used for creating new machines, disks, NICs and making reservation.

#### Input Controls

The input control in this web interface provides the necessary components that supports the functionality of the software. There is array of HTML input control, but we have chosen a few that satisfy our specifications. These input controls includes text field, buttons, date fields, drop down list and list box. They are used for both input and output functions like listing the inventory, inserting texts, to update data, to send queries, viewing selection options and searching inventory. These controls are designed with HTML tags and HTML template.

### 3.2.2 Server

The server is responsible for parsing all request from the web interface to the database and command protocol. When request is made on the web interface (for instance available machine on database), the server calls the specific function of command protocol to handle this request by providing the necessary resources. Likewise the server interchange communication between the user interface and the database system for executions and requests.

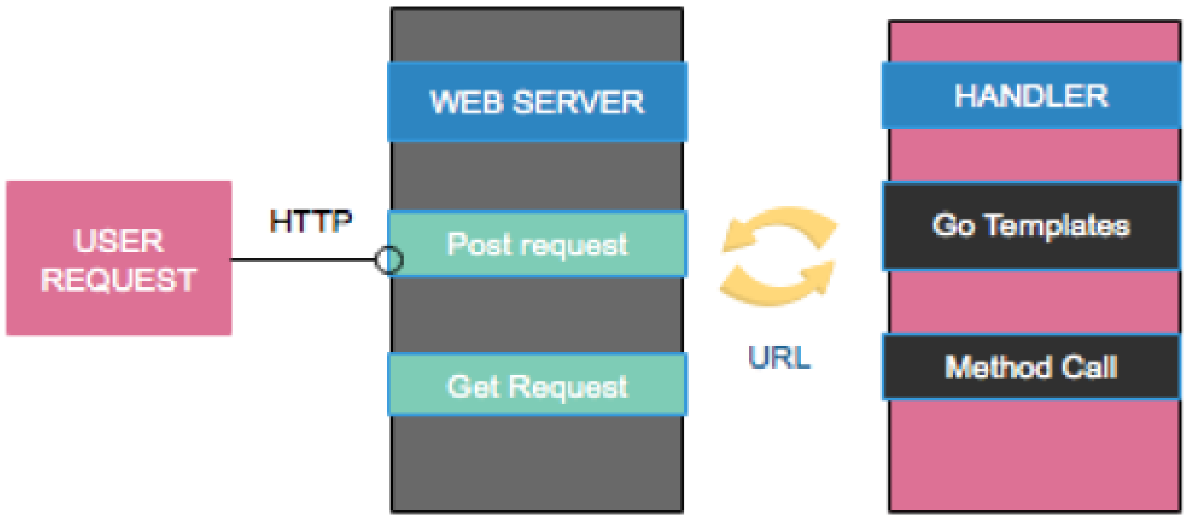


Figure 3.2: Server Activity Description

### 3.2.3 Database

The database is another major component of this system, because it is provides storage and resource for the system. The data includes reservation records, machine details, disks, and NICs information. The database is designed with SQLite model using Go programming language as the back end tech. SQlite was used here because its a fast open source SQL engine. We created a functional database system that suites the specification of the software. The database scheme contains tables of machines, user record, reservation, NICs (network interface cards) and disks.

One of the function of the software is the ability to make reservations. Below show a typical model of the database design.

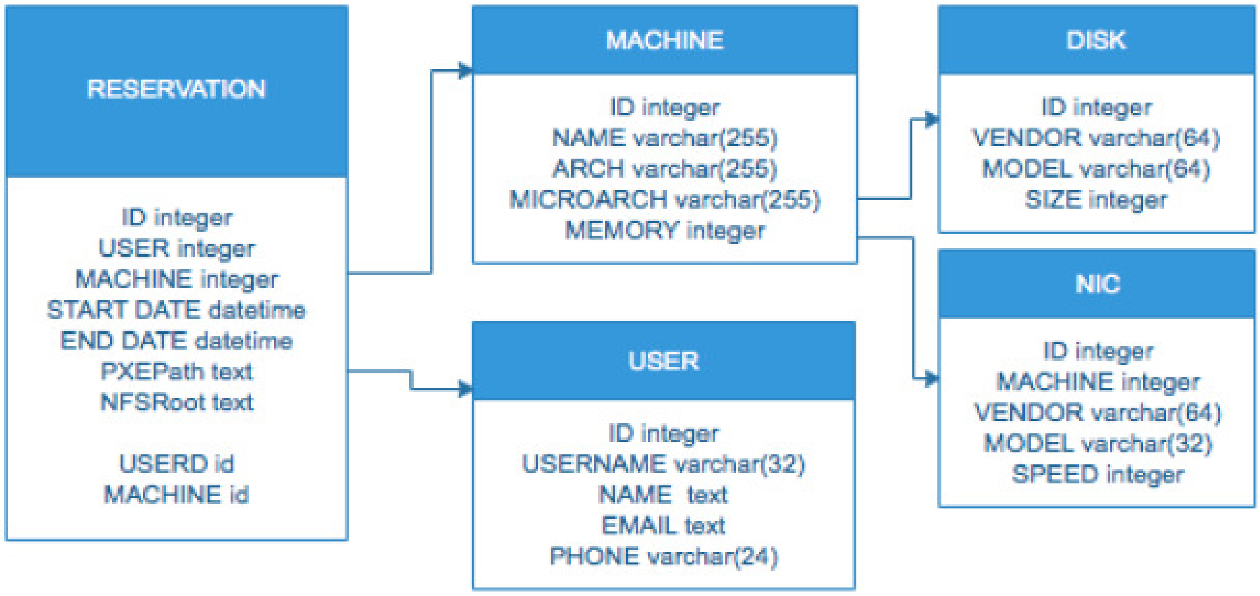


Figure 3.3: Database design

### 3.2.4 Data flow and protocols

The data flows from the user interface via the server to the database. We used Go HTTP handler to process data recieved from the HTML form to be stored in the database. The client server initiate each method (function) when the HTTP handler sends a request. For example, when users search for list of information such as machines details, the request is sent through the server as a query, and the specific Go fuction is called to fetch the data from the disegnated database. This process is the protocol that control the sequence of data flow in the system. It controls data input, handling and output. This protocol improves data circulation in the system and provides automatic command rather than manual query routin perfomed on commad line.

# 4 Implemetation

## 4.1 Review

The software implementation showcase the functionality of all the requirements as described previously. This functionality include the ability to input data and retrieve data through the system. Proper measures has been be considered to obtained maximum result as required. This measure includes the choice of tools used for the developement and the design approach. It was implemented to demonstrate the realization of the proposed specifications especialy features such as making reservations in the system.

## 4.2 Implementation tools

The user interface is implemented on the internet web browser using apeche local server port as web address. The web browser serves as the environment for testing the frontend (user interface) of the software, and while the HTML template and Go library is used for the development. The database was developed using the SQLite model schema which is coded with Go language. This platform provides robust query funtionality for creating several data schema. The HTTP server provides connections between the frontend and backend implementation. It server as a channel for parsing command protocol through user interface and database. We used local port to represent web address for this implemeting. These tools provided all the necessary component to address the requirements for this software.

## 4.3 Program Structure

The Go programming language has a fundamental developemtn structure that is categorized into packages. The packages contains a group of program file with dependencies that links them together. For this software design, we have used two main package to actualize the develpment goal. These packages inludes:

- Database package
- HTTP package

The database package contains all the Go files with database related method (struct) and functions. Each Go class or struct of hardware represented in the system depends on this package for communicationg with other classes and database resources. The HTTP package also contain some Go file relates to the user interface ( for example the funtion Handler) and HTML files. the HTML files contians several templates for the web interface content and elements. These file depends on the HTTP package for serving the web contents and post requests. Below is a table describing the list of header files in each packages.

| No | Package | Files |
|---|---|---|
| 1 | Database (pkg) | Machine.go |
| | | Reservation.go |
| | | Disk.go |
| | | User.go |
| | | nic.go |
| 2 | HTTP (pkg) | Handler.go |
| | | Server.go |
| | | Templates(.html) |

Table 4.1: Program package

## 4.4 List of Funtions

The functions contains group of statements that perform the tasks specified in the software requirement. It contains the algorithms that process different queries and schema actions. The method fields contains the entities of the hardware represented in the system, and this feilds are used as argumet in the various functions.

| No | Struct | Functions |
|---|---|---|
| 1 | machine() | initMachines() |
| | | CreateMachine() |
| | | UpdateMachine() |
| | | GetMachines() |
| | | GetAvailableMachine() |
| 2 | reservation() | initReservation |
| | | CreateReservation() |
| | | GetReservation() |
| | | FilterReservation() |
| | | SortReservation() |
| 3 | disks() | intiDisks |
| | | CreateDisks() |
| 4 | nics() | initNICs |
| | | CreateNICs |

Table 4.2: List of Funtions

## 4.5 Description of Functions

### Adding Machines to the system

Adding machines and other devices to the system is one of the software requirements. On the user interface, the software provides a text feild and forms where users can enter details of machines to be saved as data in the database. The SQLite Init funtion create a new database schema with defined table feilds where data is stored. When the user submit a AddMachine form, the InitMachine() create the database schema where the machine details is stored. It uses type (Machine) feilds as argument for creating the schema feilds. Below is the code listing for creating database:

Listing 4.1: Creating Database for machine

```
func initMachines(tx *sql.Tx) error {
        _, err := tx.Exec(`
        CREATE TABLE Machines(
                ...........
        );`)
        return err
}
```

The CreateMachine function is reponsible for parsing the details of machine to the database schema. It uses the SQLite insert query to parse a give data using the arguments to the database. When the user submit a form (AddMachine), the Go HTTP handler process this form by converting the plan text to a formdata accroding to their data types defined in the schema, and then call the CreateMachine to insert this data to the database. Below is the code list:

SPEUDOCODE:

```
1 Initialize database values to zero
2 Open the database and check for error.
3 Input values into the database.
4 If value format is ok, stored values and return.
```

Listing 4.2: Adding machines details

```
func (d DB) CreateMachine(name string, arch string,
        microarch string, cores int, memoryGB int) error {
        _, err := d.sql.Exec(`
                        INSERT INTO Machines(name, arch, microarch,
                         cores, memory)
                        VALUES (
                                      //feild values
                              )`,
                      name, arch, microarch, cores, memoryGB)
        return err
}
```

## Making a Reservation

The reservation process is similar to creating a machine but here it required the UserID and MachineID as froeign keys in creating the database schema. The UserID is used for selecting the user making the reservation, and MachineID for selecting the machine to be reserved. On the web interface (reservation page), the form has a drop down list where users can select their username and the machine they want to reserve. When users open the reservation page to create new revervation, the HTML form use the post request to get list of machines and username on the dropdown list. PSEUDOCODE:

```
1 Initialize database table to zero.
2 Open the database schema and check for error.
3 Input reservation values in the feild.
4 Select machine id from the list where machine name is chosen.
5 Select user id from the list of user where name is chosen.
6 insert all the values to reservation table and return.
```

Listing 4.3: Storing Reservation details

```
        _, err := d.sql.Exec(`
                              INSERT INTO Reservations(machine,user,start,end,
                              pxepath,nfsroot)
```

```
                    VALUES (
                                (SELECT id from Machines where name=?),
                                (SELECT id from Users where username=?),
                                //other fields
                    )`,
```

To make a reservation, users will select their username and a machine from the drop down list which is loaded by GetMachine and GetUser function. The selection query gets the UserId and the MachineId of the selected username and machine as foriegn key to the reservation database shema. Also, start and end time/date of the reservation is entered on th text feild , and this plan text is formated to match the database using a time package in Go library. After submiting the form, the Go HTTP handler handles the Insert query by taking all the arguement values and saving them to database.

## Checking Available Machines

The user can search the inventory for specific imformation. One of the example is the ability to search for available machines in the system. Available machines are those machine that are either not reserved at the moment or that are free for a certain period. This feature helps users to get list of machine free to be reserved. In this process of checking available machine, the user enters a specific start date and end date in the provided text feild, and submit the query. After submiting the query, the HTTP handler process this request, and the server calls the GetAvailableMachine method (function) to fetch this request from the database. Go HTTP handler uses a Request functon to call the server and ResponseWriter to respond to the html request [2]. The server uses ListernAndServe method to listen to incoming request and to sall the requested method to handle the request. In the GetAvalaibleMachine method, we used a SQLite sub query tech to combine the machine and reservation database table. This combination logic allows the query to scan through both table rows using reservationID and machineID to indentify the machines that are not reserved for that time range the user entered. The logic contains SQL WHERE cluase, OR and AND operator to perform comparisons between reservation dates and time. [3]The AND operator is used to allow multiple conditions in the statement, the OR operator is used to combine condition that are true, and the WHERE clause is used to specify condition while fetching data from database[5]. Below code listing shows the function

and query for this operation: SPEUDOCODE: 1 Open the database schema and check for error.
2 Query the database and select machine values where Id is not in reservation
3 Select machine values where its reserved and query date is between a start or date entered.
4 And select machine values where start or end date is between the query date.
5 Assign the selected values to the row and return the values.

Listing 4.4: Searching available

```
func(d DB) Filter_M_By_Dates(from time.Time, to time.Time) ([]Machine, error){


        rows, err := d.sql.Query(`
                SELECT m.id, name, arch, microarch, cores, memory
                FROM Machines m
                WHERE m.id NOT IN (SELECT r.machine FROM Reservations r
                WHERE (? BETWEEN r.start AND r.end)
                OR (? BETWEEN r.start AND r.end)
                AND ( r.start BETWEEN ? AND ?) OR (r.end BETWEEN ? AND ?)


        );
        `, from, to, from, to, from, to)
```

## Updating Machines details

As part of the software requirements, users can change the information stored in the database. This is done on the wen interface using edit box that is attached to every feild with that requirement. Example is updating the information of a particular computer or disks when the hardware is upgraded. In this case, the UpdateMachine method performs this action by using the SQLite update query. This update query opens the database schema of the machine and replaces the existing data with a new one. Below code listing shows the update function:

Listing 4.5: Function for Updating data

```
_, err := d.sql.Exec(`
                UPDATE Machines
                SET arch = ?, microarch = ?, cores = ?, memory = ?
                WHERE `+row+` = id


`,arch, microarch, cores, memory)
```

## 4.6 Get Funtions

The Get function is a mothod used for fetching list of machines and other devices stored in the database. Each time the user opens the web interface, the HTTP handler send server a request and server calls the GetMachines method to fetch the data from database. This information is listed in the inventory on the web interface through the HTTP ResponseWriter. The Get futction uses the SQLite selection query to scan throught the database and select the required data. Another usage for this Get function is fetching the details of a machine when the user click on a particular machine. It uses a SELECT query and WHERE condition in the logic, to speciy the machine clicked used its unique ID. These mehtods is applicable to other classes (type) such as, disks, nices and reservations.

## 4.7 Problems Encounted

During the implemetations, we encountered problems such as formating plain text to database data types, and combining two database table for search queries. We solve this query proplem by using SQL join function, which normally open two database and scan through them same time to compare different information.

# 5 Evaluations

## 5.1 Test cases

We have evaluated this software by testing different functionality as specified in the requirement. This was achieved by creating multiple cases that proves the performance of the software. This evalution was performed in real time and directly online. We have inlcuded images and table showing the test cases and results.

## Checking Available Machines

In the software specification, it is designed to have search option on the user interface. One of this search requirement is to check for available machines (unreserved machines). In the below table is list of machines reserved for different date. The test is to enter a search query with a specific date interval and ask the software to provide any machine available withing that date.

| No | Machines | Aug | Sep | Oct | Nov | Dec | Jan |
|---|---|---|---|---|---|---|---|
| 1 | Machine A | free | free | free | free | free | free |
| 2 | Machine B | free | free | free | 1st to 30th | free | free |
| 3 | Machine C | 17th - | - | - | 24 | free | free |
| 4 | Machine D | 25 | - | 25 | free | free | free |
| 5 | Machine E | free | free | 1st to 31st | free | free | free |
| 6 | Machine F | free | free | free | free | 2nd to 31st | free |

Table 5.1: List of Reservations

| No | Date: start — end | Available machines |
|---|---|---|
| 1 | 12-08-2017 to 30-08-2017 | A B E F |
| 2 | 17-08-2017 to 24-08-2017 | A B D E F |
| 3 | 01-08-2017 to 05-10-2017 | A B F |
| 4 | 05-08-2017 to 10-08-2017 | A F |
| 5 | 01-08-2017 to 30-08-2017 | A D E |
| 6 | 01-01-2018 to 30-01-2018 | A B C D E F |

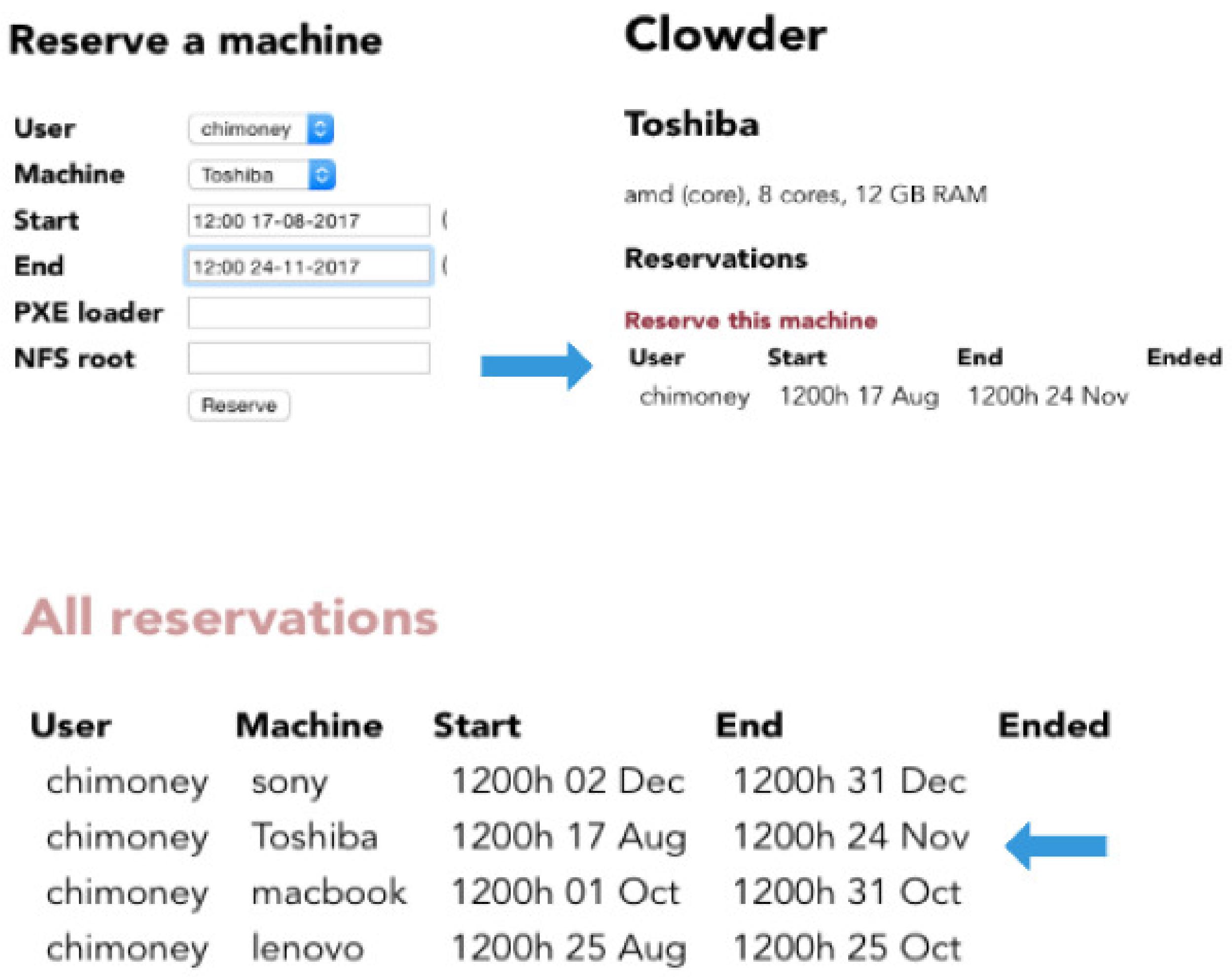
Table 5.2: test 1

| No | Date: start — end | Available machines |
|---|---|---|
| 1 | 01-08-2017 to 01-01-2018 | A |
| 2 | 01-08-2017 to 15-11-2017 | A F |
| 3 | 01-12-2017 to 30-01-2018 | B C A D E |
| 4 | 30-11-2017 to 30-12-2017 | A C D E |
| 5 | 02-10-2017 to 02-08-2017 | A D E |

Table 5.3: test 2

## Reserving a machine

Reserving machine is another requirement we have implemented in this software. This is the process where users choose and reserve a machine for definit time and date. When a reservation is made, the details of the reservation appears under the machine being reserved. Figure **??** shows the evaluting of this function, where we put details of reservation, by selecting a username, machine and inserting start and end time/date. After submiting with the reserve button, the it appears on the inventory with other reservations list.



Reserve a machine

| | |
|---|---|
| User | chimoney |
| Machine | Toshiba |
| Start | 12:00 17-08-2017 |
| End | 12:00 24-11-2017 |
| PXE loader | |
| NFS root | |

Clowder

Toshiba

amd (core), 8 cores, 12 GB RAM

Reservations

| User | Start | End | Ended |
|---|---|---|---|
| chimoney | 1200h 17 Aug | 1200h 24 Nov | |

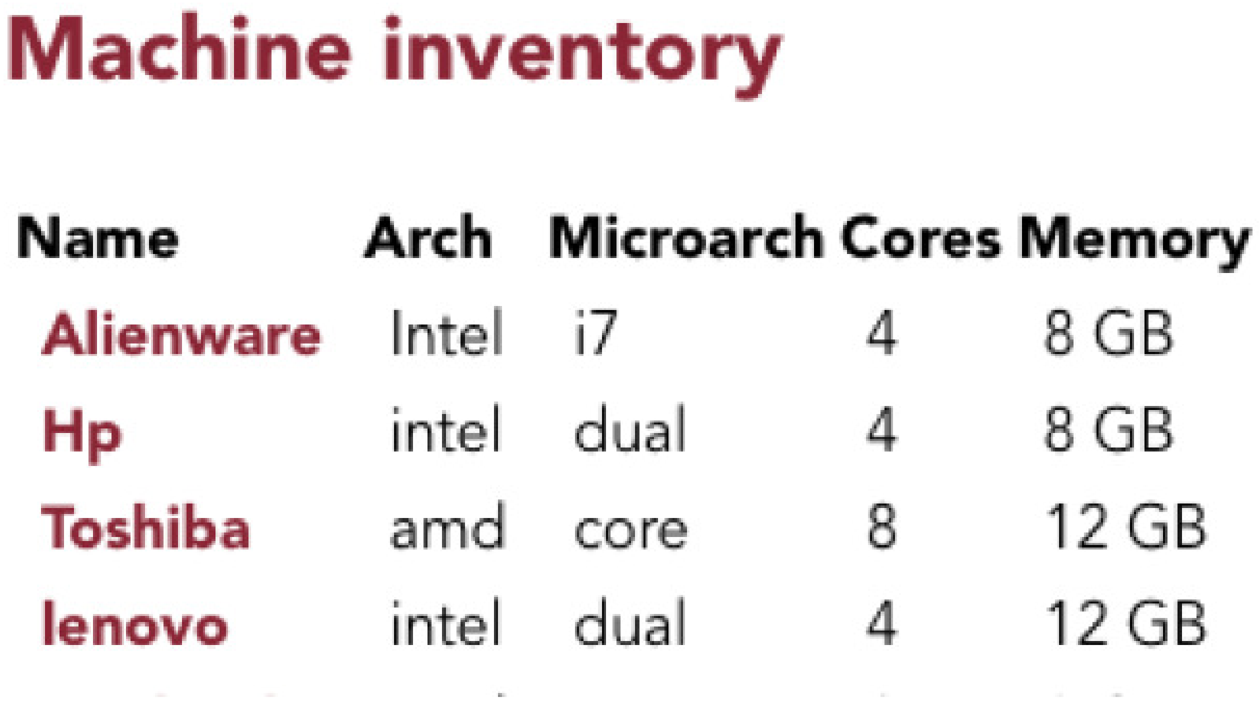
All reservations

| User | Machine | Start | End | Ended |
|---|---|---|---|---|
| chimoney | sony | 1200h 02 Dec | 1200h 31 Dec | |
| chimoney | Toshiba | 1200h 17 Aug | 1200h 24 Nov | |
| chimoney | macbook | 1200h 01 Oct | 1200h 31 Oct | |
| chimoney | lenovo | 1200h 25 Aug | 1200h 25 Oct | |

Figure 5.1: Reservation function

## Updating a machine

Update functionality allow users to change the details of a machine already stored in the database. This is to enable them update the information about each machine as needed when ever there is an upgrade in the system. We have tested this functionality by changing the entire information of a machine listed in the inventory. Figure ?? shows an example of this test case.



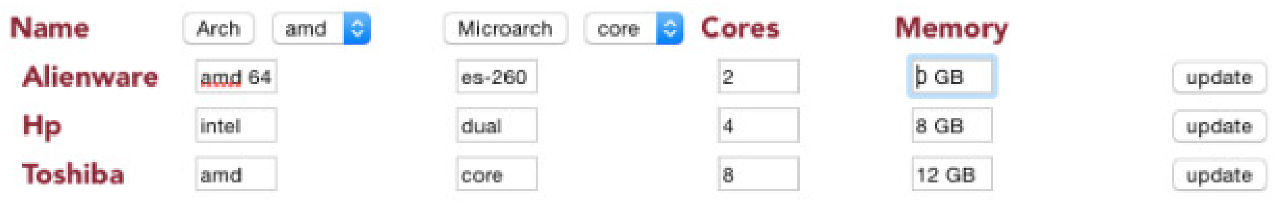
Machine inventory

| Name | Arch | Microarch | Cores | Memory |
|---|---|---|---|---|
| Alienware | Intel | i7 | 4 | 8 GB |
| Hp | intel | dual | 4 | 8 GB |
| Toshiba | amd | core | 8 | 12 GB |
| lenovo | intel | dual | 4 | 12 GB |

Figure 5.2: Before update

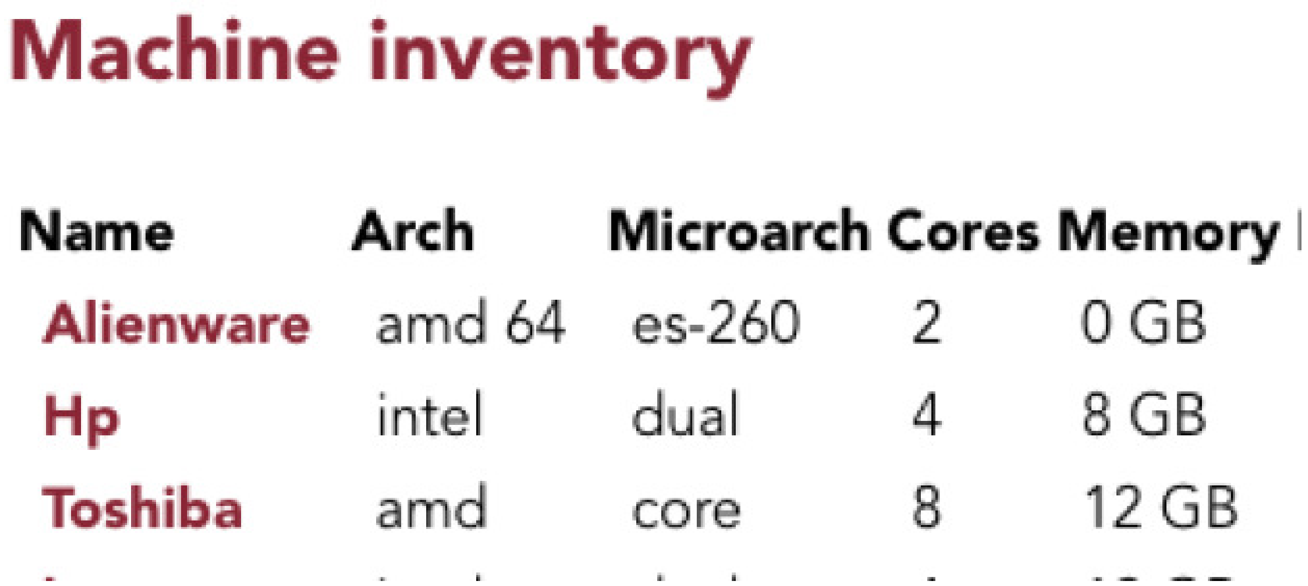
## filtering inventory

This functionality helps user to filter the inventory using the context of desired information. User is able to filter the machines inventory with memory size. The size range is inserted in the search to get the list of machines that fall within the range. When the button is clicked, the server returns the list of machine that falls in the range. Another type of filtering is the the drop list filter. This type has a dropdown menu user select a context option to search for. For exaple, user can filter the machine inventory by selecting type os microarchitectre and architect..



| Name | Arch | amd | Microarch | core | Cores | Memory | |
|---|---|---|---|---|---|---|---|
| Alienware | amd 64 | | es-260 | | 2 | 0 GB | update |
| Hp | intel | | dual | | 4 | 8 GB | update |
| Toshiba | amd | | core | | 8 | 12 GB | update |

Figure 5.3: Editing machine details

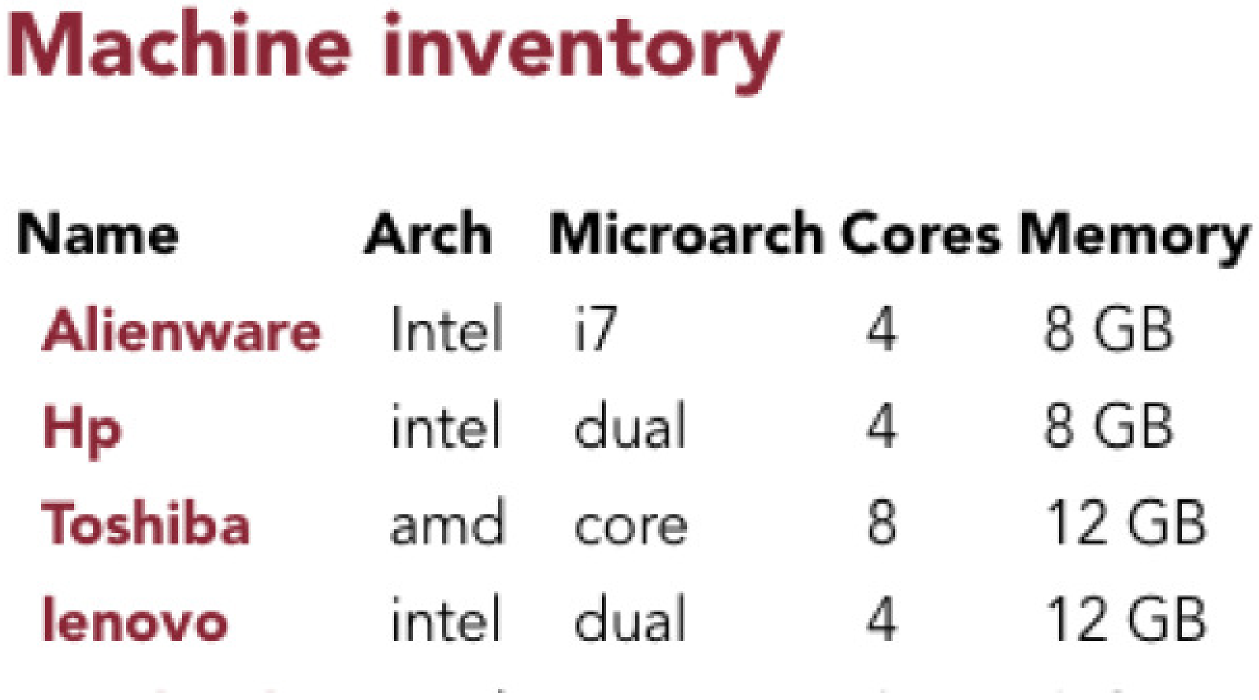


Machine inventory

| Name | Arch | Microarch | Cores | Memory |
|---|---|---|---|---|
| Alienware | amd 64 | es-260 | 2 | 0 GB |
| Hp | intel | dual | 4 | 8 GB |
| Toshiba | amd | core | 8 | 12 GB |

Figure 5.4: Updated machine properties

## 5.2 Evaluations



Machine inventory

| Name | Arch | Microarch | Cores | Memory |
|---|---|---|---|---|
| Alienware | Intel | i7 | 4 | 8 GB |
| Hp | intel | dual | 4 | 8 GB |
| Toshiba | amd | core | 8 | 12 GB |
| lenovo | intel | dual | 4 | 12 GB |

Figure 5.5: Reservation function

# 6 Conclusion

The need to complete the clowder system which was design to manage test machines has been accomplished by addressing the issues of system managemnt and accessibility. It was required to have a flexible user interface for dynamic accessibility, and a functional database system for storage and management. This issues has been resolved by designing a frontend and extended backend software that provides the required functionality for the user interface and the database. We have created a web interface where users can communicate with the test by sending request via HTTP protocol. This web interface contains the required functionality for making reservations, updating machine details and viewing the inventory and generally resolve the issue of user accessibilty. Also, as part of the requirement to complete the Clowder system, we created a SQL database for storing and retrieving data. This database enables the clowder system to manage the access and users activity on the test machines by storing their details and recording all reservations made by users. The the database we have created has the ability to plug in a DHCP server which provides network between the Clowder and the test machines. All these factors has made is easier for researchers to run their test on the machines without conflict and lost of data.

# Bibliography

[1]

[2]

[3]

[4]

[5] S. W. Clause.

[6] M. Johnston. Preboot execution environment (pxe) specification, 1999.

[7] M. LLC.