Clowder: Software for Managing Clusters Of High Performance Computers

by

© *Samson Ugwuodo*

A Project Report submitted to the

School of Graduate Studies

in partial fulfillment of the

requirements for the degree of

Master of Science

Scientific Computing Program

Memorial University of Newfoundland

*N*ovember 2017

St. John's Newfoundland

Abstract

Clowder is a system designed to help researchers in the Faculty of Engineering and Applied Science to manage and control clusters of test machines. Some of the features that required improvement in this system were the ability to reserve a machine for a certain period of time, a function that allows users to add more computers and manage their properties, and 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 improves the 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 approach were 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 Applied 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

List of Tables

List of Figures

1 Introduction

The use of high performance computers for research in the Faculty of Engineering and Applied Science has increased, because of article increase in research tasks, such as testing new operating systems. As researchers demand robust computers, these computers systems also need to be expanded to accommodate these large tasks. As we expand the systems, access and usage becomes more complicated to manage. Therefore we are faced with issues of system management, system accessibility, and storage. These issues are the inability to provide a proper record of each computer system and its usage, the inability to provide flexible user 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 necessary to develop software that solves these problems for the Faculty of Engineering and Applied Science.

Clowder is a system designed to provide a Preboot Execution Environment (PXE) for testing new operating systems through a Network File System (NFS) sever. Others have worked on this system for several years, but there was more work yet to be done to improve its existing features and functionality. As an example, it takes 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 the current manual, error-prone norm (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, making it a complete software tool.

We have accomplished this goal by using a design approach that augments a previously existing interface and database system. 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 provides users with the

system inventory and other user activities. This software provides the ability to search for data in the inventory, and to add new computer systems 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 to end reservations as well.

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 microarchitecture is stored in the database. The 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 this information stored in the database tables serves as input data for the program. 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. This interface can also partially replace the SSH-accessible command line prompt which was the previous user interface for Clowder system. This choice provides flexible access to and control of the system, by allowing users to log on to the system and make requests of the inventory at any time through the web server. Figure 1.1 shows a general description of the Clowder system.

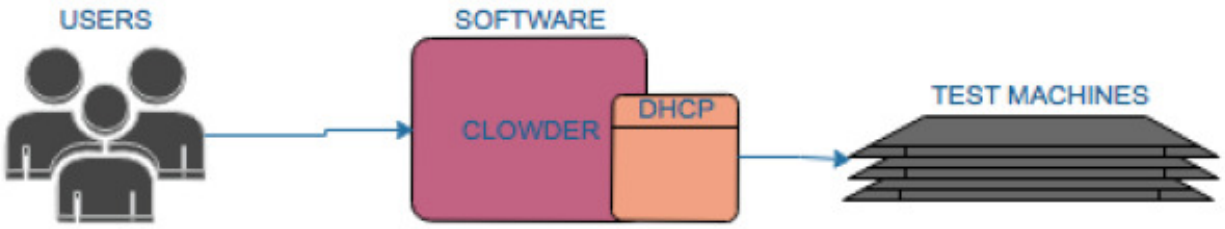

Figure 1.1: A general design of the system

This figure showcases the background concept of the Clowder as a full system. This concept is shown in detail in Fig 3.1, which explains 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 components that made up the design.

1.1 Contribution

I have contributed to this project to improve its existing interface and database functionality in other to have Clowder as complete software tool. My work includes the addition of required features to the interface such as; searching data from the system inventory, making reservation, checking for available machines, updating the system data, filtering and sorting the system inventory. These features were extended to the database improve its existing functionality. I have further explained this contributions in section 3.2 and section 4.3.

2 Background

2.1 Clowder

Clowder is made up of various components, and some of these components have been worked on by researchers in the department. The 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 improved the existing user interface by adding a dynamic web interface which is extended to the back end with more functionality. 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.

2.2 PXE and DHCP

PXE is a standard Internet protocol widely deployed using DHCP which help to distribute IP. This process forms a DHCP conversation 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. The operations of standard DHCP servers that serve up IP addresses will not be disrupted by the use of the extended protocol [4]. 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 [4]. These files are managed with a Network File System (NFS) and TFTP, which allow

users to access files across networks. This processes serves as the main function of Clowder system as a testing tool.

2.3 SQL

SQL is a special purpose programming language designed for managing information in a relational database management system (RDBMS). For example it can used to record information about an organization and their activities. So by using a relational database, you can save this information as two tables that represent two distinct entities: organization and activities. Here information about an organization and activities is stored in two different tables with unique identifications (ID). In the design of some front end functionality extended to back end, we used a SQL JOIN query for combinations of different database data where our algorithms requires to merge multiple data for some query request. A SQL JOIN is a Structured Query Language (SQL) instruction to combine data from two sets of tables [7]. To associate them together, the entity is described with a primary key referring to its ID and a foreign key referring to the other table. SQL JOIN is applied by joining two entities tables using the relationship established with the foreign key. There are different types of join query which includes INNER: for collecting records that have matching values of two tables, OUTER: for collecting all records matching from either left or right tables, and RIGHT: for collecting records from right table and match with the record of other tables. This concept was used in this project to accomplished some of the functionality we have specified.

2.4 SQL Injection Attack

SQL Injection is a security threat that allows attackers to obtain unauthorized access to a databases with the goal to manipulate or modify the data in the database. A web application that lacks appropriate authentication or privileges will be vulnerable to attacks, and create a gateway for attackers to lunch their attacks on the database. The techniques used by attackers includes;

identifying injectable parameters, performing database finger printing and determining database schema. These techniques helps the attacker to get necessary information which can compromise the integrity of the database security[2]. An example of SQL injection attack is a Tautology based attack. In this type of attack, the attacker injects a malicious code (eg.'OR'1='1) into the condition statement which will always evaluate true[2]. This type of attack if successful, helps the attacker to bypass the authentication page and have access to obtain data. In the project, we have addressed this issue by putting into consideration some necessary prevention measures. An example includes; avoiding use of string concatination in the query statement and using appropriate placeholder like (where name = ?) in parameterized queries. The idea of user authentication is another prevention measure considered, but authentication was not implemented as its not part of project design.

2.5 Go Struct and Methods

In Go programming language a struct replaces the concept of class as used in other Object Oriented Programming languages. Go support the concept of methods like Java for example. But Java methods are defined within a class, while Go methods are associated with a struct. A struct is a type that contains some fields[12]. These fields have data types that represent their values. We can initialize this struct by creating an instance of it inside other functions [1]. The method is a function with name that has a receiver argument. In this project, I used the struct to create type of machine, reservation, disk and NIC. Inside this struct we have several methods that contains the algorithms that process different queries and schema actions. The method contains difference receiver types that represent the entities of the hardware in the system, and are used as arguments in the various functions.

3 Design and Requirements

3.1 Requirements

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

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

3.1.1 Data Inventory

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 machines in the system, this feature is responsible for providing the information already on the web interface. We have the ability to access this inventory faster by adding some functionality in the interface. Example is the ability to sort the inventory chronologically or by data sequence.

3.1.2 Searching Inventory

As there are lots of different data listed in the inventory, it is necessary to have a function that allows users to search for information in the inventory. Searching the inventory helps the users to get specific data by typing in queries such as dates, times, NFS root, PXE and memory sizes of a particular machine or reservation. 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 search for available machines, and therefore allowing the user to know which machine is free for reservation.

3.1.3 Make 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 reserve a 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.

3.1.4 Add and Update Data

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 validate this changes in the database.

3.2 Design

The design structure comprises the various segments and element of the user interface (front end) combined with the database (back end). In this design some of the components such as the database

structure has been worked on by others, so I added more functionality during the implementation that contributed to the requirements. I worked on methods that helps in searching for unreserved machines, sorting data, updating data and filtering inventory, making reservations and creating new Disks and NICs. Figure 3.1 describes the main structure of the design in terms of operation. We augmented the web interface that provides flexible and dynamic access to the system with

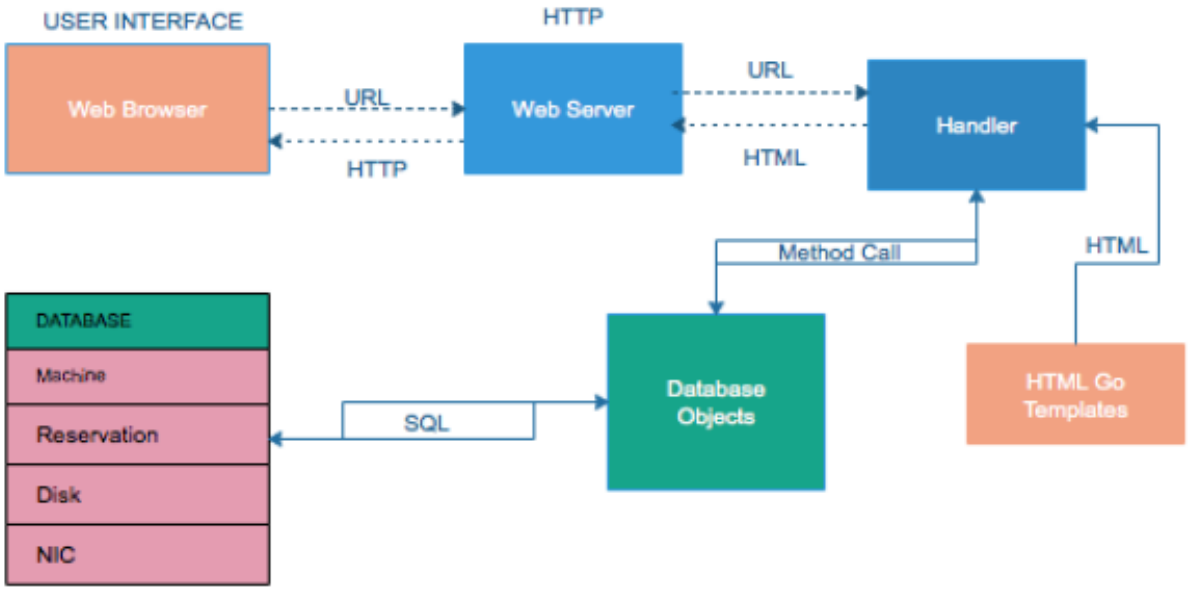

Figure 3.1: The Design

interactive functionality. This web page contains inputs and output features. The reason for choosing a web page instead of other options is to provide online dynamic and simultaneous access to the system rather than performing manual command prompts. Another reason is to have a functional user interface that present the activities of the Clowder. The web server is responsible for serving the web page with all the resources requested by the users, and it serves as a channel between the user and the database. 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 web page the server will pass this information to the database which is controlled by the main program.

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 for the front end and Go programming language for the back end. The web interface contains all the necessary elements required to have dynamic and flexible access to the system. These elements include Forms and Input controls.

Forms

Forms are one of the elements that provides the ability to input data to the database. The data is submitted through a HTTP POST request [8]. 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 HTTP form, the Go HTTP framework in the web server call a registered 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 are numerous HTML input control, but we have chosen a few that satisfy our specifications. These input controls include text fields, 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 Web Server

The web server is a program responsible for parsing all requests from the web interface using HTTP. When a request is made on the web interface (for instance querying available machines), the server calls the a function to handle this request by providing the necessary resources from the database. And interchanging communication between the user interface and the database system for executions and requests. For example, when user create a new reservation which requires to open a new page, our server will serve the web page with the desire HTML template and updated data as requested.

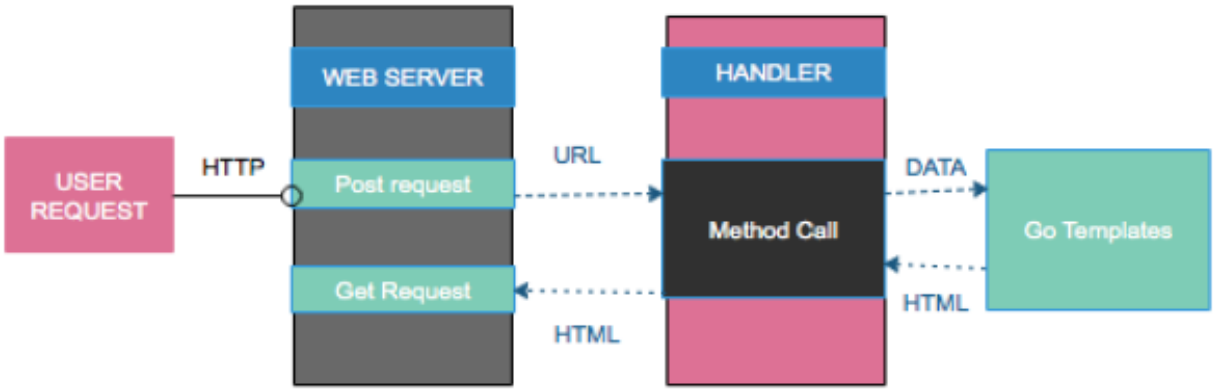

Figure 3.2: Server Activity Description

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 (network interface card) information. The database is designed with SQL model using Go programming language as the back end tech. SQL was used here as prototype because it is fast and easy to set up. We created a functional database system that address the specification of the software. The database scheme contains tables of machines, user record, reservation, NICs and disks. One of the functions of the software is the ability to make reservations. Figure 3.3 shows a typical model of the database design.

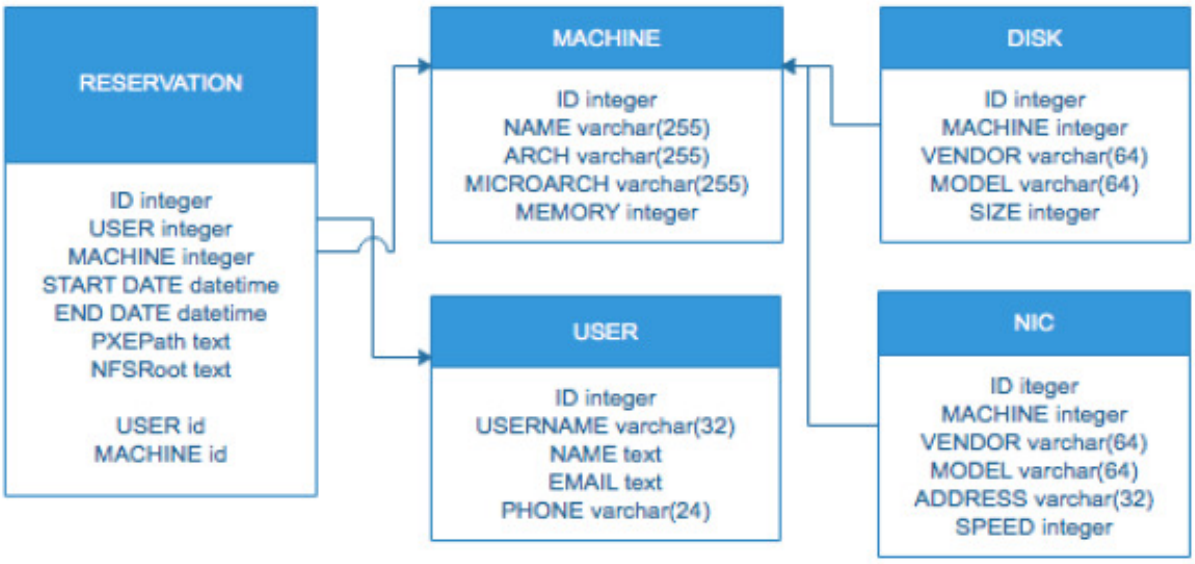

Figure 3.3: Database design

3.2.4 Data Flow

The data flows from the user interface via the web server to the database. We used Go HTTP handlers to process data received from the HTML form to be stored in the database. The server initiates each method (function) when the HTTP handler sends a request. For example, when users search for list of information such as machine details, the request is sent through the server as a query, and the appropriate Go function is called to fetch the data from the designated database. Go HTTP handler uses a Request functon to call the server and ResponseWriter to respond to the HTML request [6]. The server uses ListenAndServe method to listen to incoming request and to call the requested method to handle the request. This process controls the sequence of data flow in the system. It controls data input, handling and output. This protocol helps the dynamic functionality on the user interface to perform in the system and also provides automatic command rather than manual query routine performed on command line. Listing 3.1 shows example of handler function and Listing 3.2 shows the registered function in the server.

Listing 3.1: Sort Memory function in the handler

```
1 func (s Server) sortmPage(w http.ResponseWriter, r *http.Request) {
2     s.logRequest(r)
3     tname := "machines.html"
4     t, err := getTemplate(tname)
5     if err != nil {
6         s.Error(err)
7         templateError(w, tname, err)
8         return
9     }
10    sortmemory,err := s.db.Sort_Memory()
11        if err != nil {
12        s.db.Error(err)
13        renderError(w, "Error sorting",
14            fmt.Sprintf("Unable to sort machines: ",
15                err))
16        return
17    }
18    t.Execute(w, sortmemory)
19 }
```

Listing 3.2: Calling Sort Memory function

```
1 http.HandleFunc("/machines/Sort_Memory/", server.sortmPage)
```

4 Implementation

4.1 Review

The software implementation showcases the functionality of all the requirements as described in section 3.1. This functionality includes the ability to input data and retrieve data through the system. We have accomplished this task by implementing the designed structure. This include the choice of tools used for the development and the design approach. It was implemented to demonstrate the realization of the proposed specifications.

4.2 Implementation tools

The user interface is implemented on the internet web browser using a HTTPS port. The web browser serves as the environment for testing the front-end (user interface) of the software, while the HTML templates and Go library are used for development. The database was developed using the SQL model schema. The Go HTTP server provides connections between the front-end and back-end implementation. It serves as a channel for parsing requests through the user interface and database. These tools provided all the components necessary to address the requirements for this software.

4.3 Program Structure

The Go programming language has a development structure that is categorized into packages, see Table 4.1. The packages contains a group of program file with dependencies that link them together. For this software design, we have written two main packages to actualize the development goal. In addition to other people's works in these packages, we have added more files with numerous

methods and functions of couple or more thousand lines of SQLite, Go and HTML codes that improves the system requirements. These packages include:

- Database package
- HTTP package

The database package contains all the Go files with database-related structs. The Go objects represent the same data as in the database. And each Go struct depends on this package as the object-relational mapping tool (ORM) between other structs and database resources. The HTTP package contains Go file related to the user interface (for example the function handlers) and HTML files. The HTML files contains several templates for the web interface content and elements. These files depends on the GO HTTP package for serving web contents and post requests. Table 4.1 describes the list of files in each package.

No	Package	File
1	Database (pkg)	machine.go
		reservation.go
		disk.go
		user.go
		nic.go
2	HTTP (pkg)	handler.go
		server.go
		frontpage.html
		machine.html
		reservation.html

Table 4.1: Program packages

4.4 List of Struct and Methods

Figure 4.1 shows the class diagram of the list of struct and methods with their relationships with others. It shows the group of methods that perform the tasks specified in the software requirement in our design.

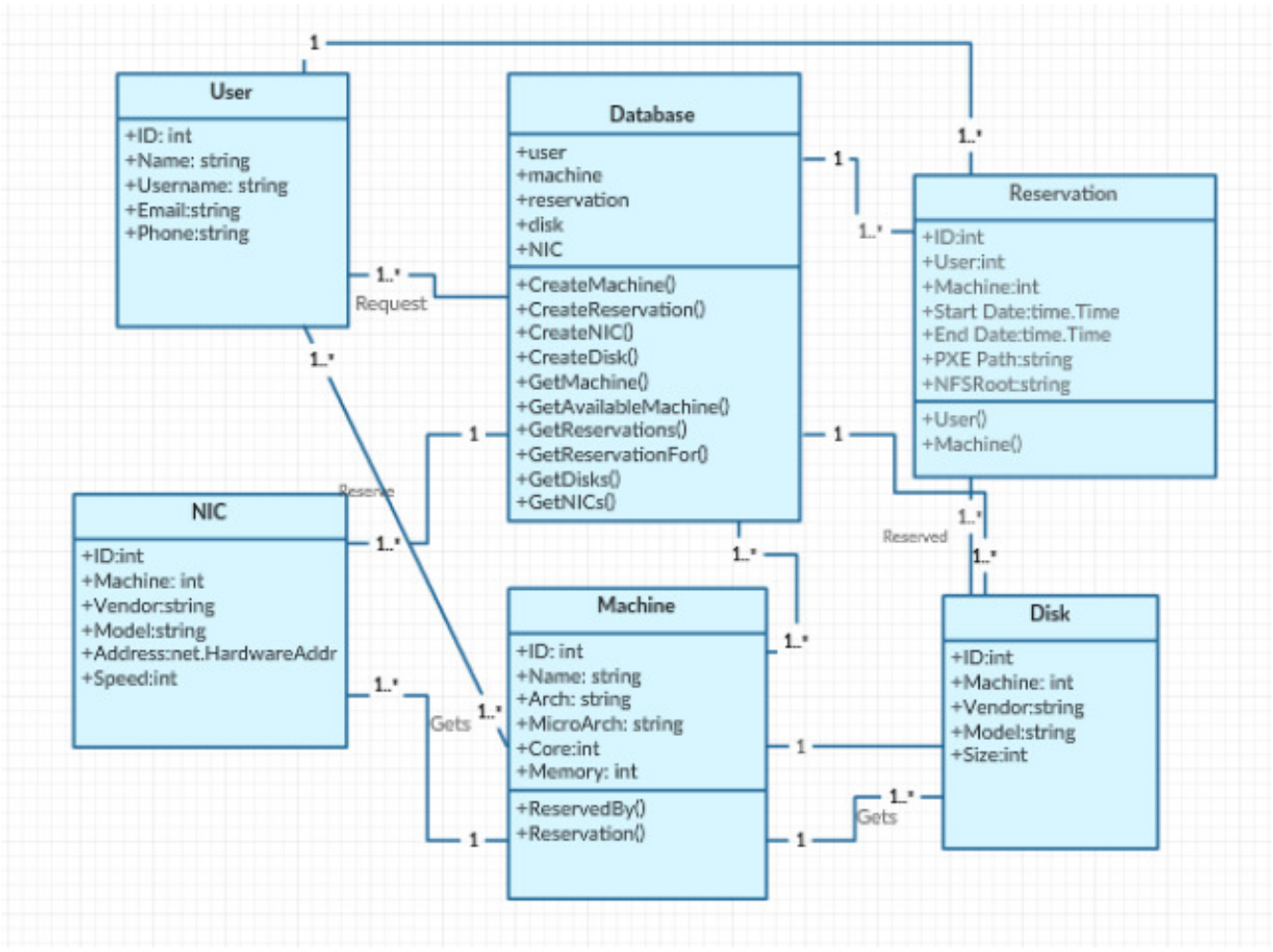

Figure 4.1: Class Diagram showing struct and method relationships

4.5 Implementation of Required Functionality

Adding Machines to the system

Adding machines and other devices to the system is one of the software requirements in section 3.1.4. On the user interface, the software provides a text field and forms where users can enter details of machines to be saved as data in the database. The initMachines method creates a new database schema with defined table fields where data is stored. When the user submits a Add Machine form, the data is stored in the table created by initMachines. Listing 4.1 is the code listing for creating the database:

Listing 4.1: Creating table for machines

```
1 func initMachines(tx *sql.Tx) error {
2     _, err := tx.Exec(`
3     CREATE TABLE Machines(
4         ...........
5     );`)
6     return err
7 }
```

The CreateMachine method is responsible for inserting data to the database schema. It uses the SQLite INSERT query to parse the given data using the arguments to the database. When the user submits a form (AddMachine), the Go HTTP handler processes this form by converting the plain text to data according to their data types defined in the schema, and then calls CreateMachine to insert this data in the database. Listing 4.2 shows the code list.

Listing 4.2: Adding machines details

```
1 func (d DB) CreateMachine(name string, arch string,
2     microarch string, cores int, memoryGB int) error {
3     _, err := d.sql.Exec(`
4             INSERT INTO Machines(name, arch, microarch,
5              cores, memory)
6             VALUES (
7                 //field values
8             )`,
9         name, arch, microarch, cores, memoryGB)
10     return err
11 }
```

Making a Reservation

The reservation process is similar to creating a machine but it requires the User ID and Machine ID as foreign keys in creating a reservation 3.1.3. The User ID is used for selecting the user making the reservation, and Machine ID 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 user name and the machine they want to reserve. When users open the reservation page to create new reservation, the HTML form use the post request to get list of machines and user name on the drop-down list.

Listing 4.3: Pseudocode

1. Begin transaction.
2. Select machine and user id from the list where name is chosen and input values.
3. Check for conflicting input and rollback..
4. Insert data if conflict doesn't exist.
5. Commit transaction and return result.

Listing 4.4: Storing Reservation details

```
1     ('BEGIN;
2           INSERT INTO Reservations (machine,user,start,end,pxepath,nfsroot)
3           SELECT (SELECT id from Machines where name=?),(SELECT id from Users
              where username=?),?,?,?,? where not exists (select r.id from
              Reservations r
4           INNER JOIN Machines m
5           ON m.id = r.machine where m.name = ? AND (? >= start AND ? <= end) OR
              (? <= start AND ? >= end));COMMIT;',machine,user,start,end,pxepath,
              nfsroot,machine,start,start,end,end)
```

To make a reservation, users select their user name and a machine from the drop down list which is loaded by GetMachine and GetUser method. The selection query gets the User ID and the Machine ID of the selected user name and machine as foreign key to the reservation database schema. Also, start and end time/date of the reservation is entered on the text field , and this plain text is formatted to match the database using a time package in Go library. After submitting the form, the Go HTTP handler handles the Insert query by taking all the argument values and saving them to database.

Concurrency Issues

Concurrency issue is the condition where two or more threads want to write data into the same storage location in database at same time. It can cause lost of data or affect the integrity of records when one data overrides another entry. In our design, this can occur when there is a conflict between two reservations. Example is when two users want to reserve the same machine with overlapping or same dates. SQLite initiate a transaction for each process to make sure that two processes doesn't access the database in incompatible manner at same time . We implemented a logical condition that checks for data conflict before a reservation process is completed. The logical condition compares the data (date/time) entered with the existing data, and if they conflict or overlaps each other then that reservation process will abort.

Checking Available Machines

The user can search the inventory for specific information. One example is the ability to search for available machines in the system as per requirement 3.1.2. Available machines are those machines 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 machines, the user enters a specific start date/time and end date/time in the provided text field, and submits the query. After submiting the query, the HTTP handler processes this request, and the server calls the GetAvailableMachine method of the

machine to fetch this request from the database. In this method, we used a SQLite JOIN sub query to combine the machine and reservation database table. It helps to indentify the machines that are not reserved for that time range the user entered. The logic contains SQL WHERE, OR and AND operator to perform comparisons between reservation dates and time. [11]The AND operator is used to allow multiple conditions in the statement, the OR operator is used to combine conditions that are true, and the WHERE clause is used to specify condition while fetching data from database[5]. Listing 4.5 shows the function and query for this operation.

Listing 4.5: Searching available

```
1 func(d DB) Filter_M_By_Dates(from time.Time, to time.Time) ([]Machine, error){
2          rows, err := d.sql.Query(`
3         SELECT m.id, name, arch, microarch, cores, memory
4         FROM Machines m
5         WHERE m.id NOT IN (SELECT r.machine FROM Reservations r
6         WHERE (? BETWEEN r.start AND r.end)
7         OR (? BETWEEN r.start AND r.end)
8         AND  ( r.start BETWEEN ? AND ?) OR (r.end BETWEEN ? AND ?)
9
10    ); `, from, to, from, to, from, to)
```

Updating Machines details

As part of the software requirements 3.1.4, users can change the information stored in the database. This is done on the web interface using text box that is attached to every field with that requirement. An example is updating the information of a particular computer or disk 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. Listing 4.6 code listing shows the update function.

Listing 4.6: Function for Updating data

```
1     _, err := d.sql.Exec(`
2             UPDATE Machines
3             SET arch = ?, microarch = ?, cores = ?, memory = ?
4             WHERE `+row+` = id
5     `,arch, microarch, cores, memory)
```

4.6 Get methods

The Get methods is used for fetching lists of machines and other devices stored in the database. Each time the user opens the web interface, the HTTP handler sends the server a request and the 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 Response-Writer. The Get methods use the SQLite selection query to scan through the database and select the required data. Another usage for Get methods is fetching the details of a machine when the user clicks on a particular machine. A SELECT query and WHERE condition are used in the logic, to specify the machine clicked used its unique ID. This concept is applicable in other struct (type) such as, Disks, NICs and Reservations.

4.7 Problems Encountered

During the implementation, we encountered problems such as parsing plain text to database data types. For example, some functions can parse plain text as they are to the database, but in some cases where the values have a different data type (eg. Hardware Mac address) cant be store as plain text. So this problem was solved by importing functions from the Go library to support those data types. Another issue encountered is trying to combine two database table for search queries. This problem was mostly encountered in the implementation of search query, where two tables were combined for comparing different data. We solved this query problem by using SQL JOIN function, which normally link different data from multiple database table by scanning through them. With

this problem solved, it helped us during the implementation to add more functionality to the user interface without having further error prone situations.

5 Evaluations

5.1 Test cases

We have evaluated this software by testing different functionalities as specified by the requirements. This was achieved by creating multiple cases that demonstrates the performance of the software.

Checking Available Machines

In the software specification section 3.1.2, it is required to have a search options on the user interface. One of this search requirement is to check for available machines (unreserved machines). Figure 5.2 is a list of machines reserved for different dates. The test is to enter a search query with a specific date interval and ask the software to provide any machine available within that date. We have randomly generated different machines and reservations to show an evaluation of the reservation process. Figure 5.1 shows a scatter plot demonstrating test coverage. So after the random generation of different reservations on different machines, we used a set of 5 machines with 10 reservations to evaluate another test case as shown in Figure 5.3. Here we can see the search result obtained from seven set of date searched for on this random data.

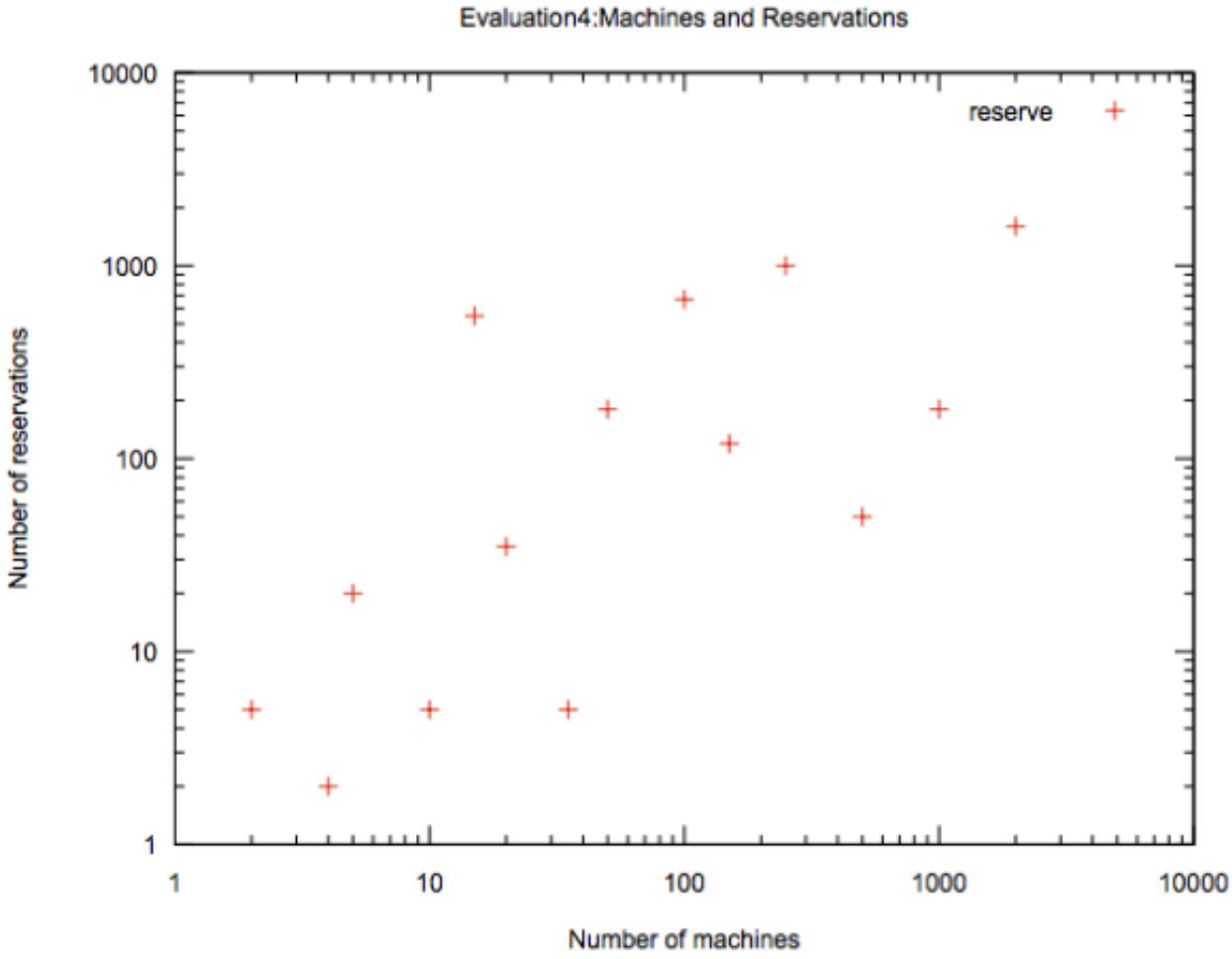

Figure 5.1: Scatter plot showing different random generated reservations

MACHINES(5)	RANDOM GENERATED DATE OF RESERVATIONS (10 reservations)
A	25-11-2017 to 06-08-2018 20-04-2017 to 16-01-2018
B	21-08-2017 to 17-07-2018 04-02-2018 to 13-02-2018 23-12-2017 to 31-01-2019
C	17-03-2017 to 09-09-2017
D	23-12-2017 to 31-01-2018 06-04-2017 to 10-01-2019
E	18-06-2017 to 3107-2017 17-11-2018 to 17-10-2019

Figure 5.2: Illustration of reserved machines and dates

SEARCHED FOR : START	END	AVAILABLE MACHINE: A	B	C	D	E
25-01-2017	25-12-2017	reserved				available
01-06-2017	01-06-2019	reserved				
30-01-2017	01-02-2017	available				
10-09-2017	10-10-2019	reserved		available		reserved
02-02-2018	11-01-2018	reserved		available	reserved	available
30-09-2019	30-01-2020	available	reserved		available	reserved
18-03-2017	25-12-2019	reserved				

Figure 5.3: Illustration of searched machines and availability

No	Date: start — end	Available machines
1	25-01-2017 to 25-12-2017	E
2	01-06-2017 to 01-06-2019	
3	30-01-2017 to 01-02-2017	A B C D E
4	10-09-2017 to 10-10-2019	C
5	02-02-2018 to 11-01-2018	C E
6	30-09-2019 to 30-01-2020	A C D
7	18-03-2017 to 25-12-2019	

Table 5.1: Test case

Reserving a machine

Reserving a machine is another requirement in section 3.1.3. This is the process where users choose and reserve a machine for a definite time and date. When a reservation is made, the details of the reservation appear under the machine being reserved. Figure 5.4 shows the evaluating process of this function, where we reserved a machine by selecting a user name, machine and inserting start and end time/date. After submitting with the reserve button, it appears on the inventory with other reservations. During testing we tried to input a wrong date and time format and we got an error as shown in Figure 5.6. This demonstrates that the software will not take an invalid date format for a reservation rather it will issue a warning to the users appropriately.

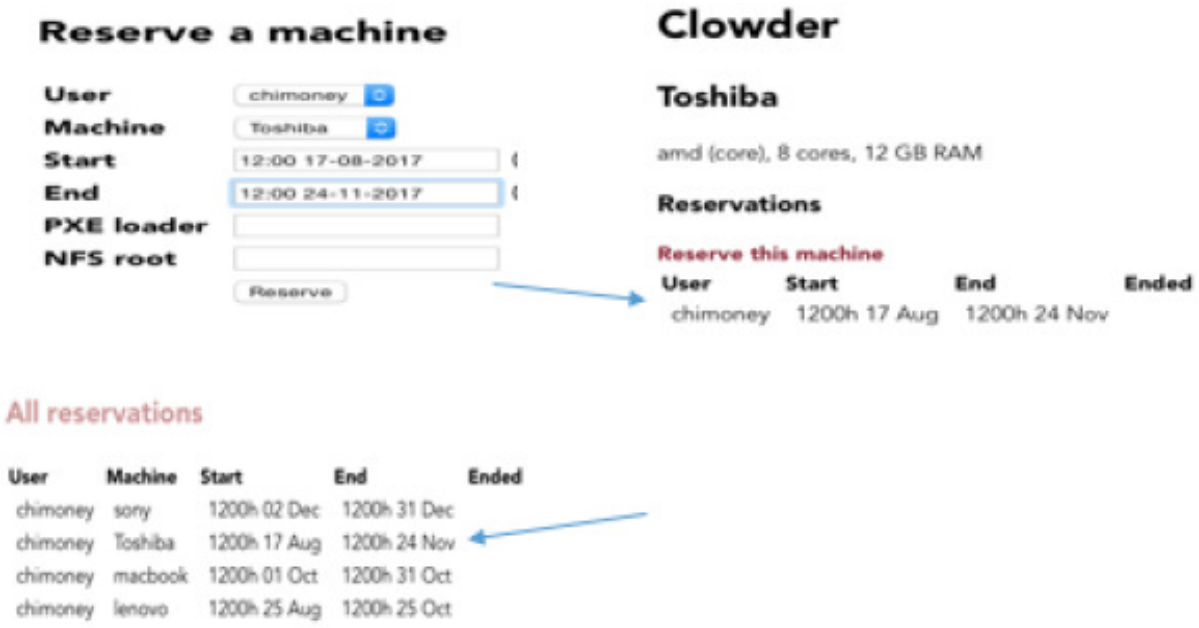

Figure 5.4: Reservation function

Updating a machine

Update functionality, as in section 3.1.4, allows users to change the details of a machine already stored in the database. This is to enable them to update the information whenever there is an upgrade in the system. We have tested this functionality by changing the details of a machine listed in the inventory. Figure 5.9 shows the process of updating the inventory. This process has demonstrate that this function performed as required because data was collected accordingly and stored without any error.

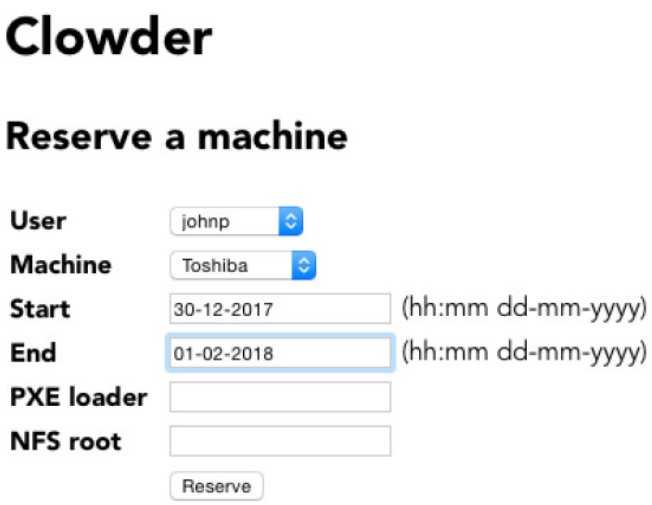

Figure 5.5: Reserving with wrong date

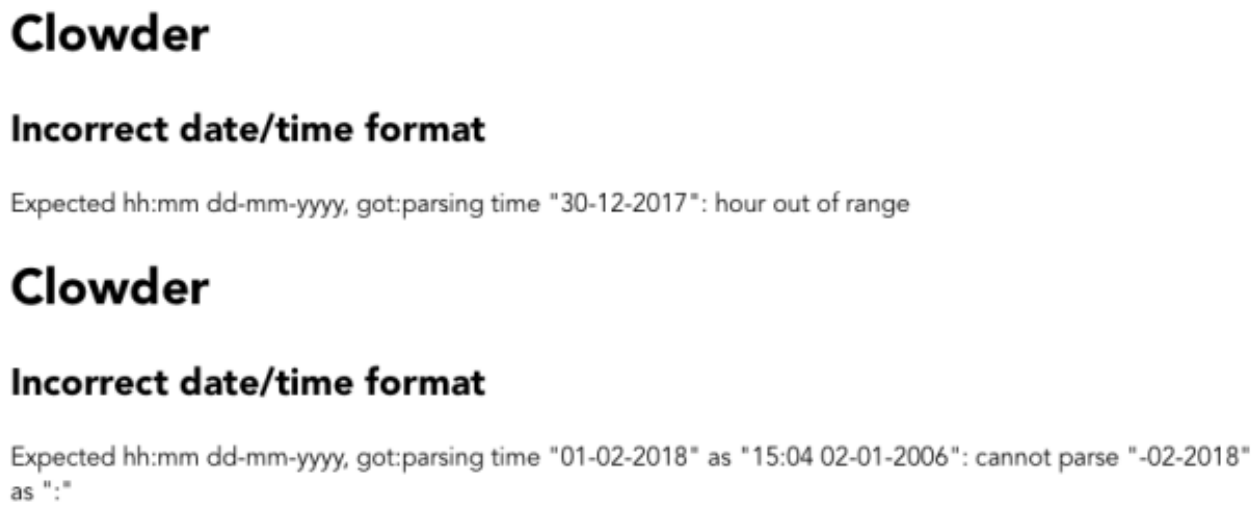

Figure 5.6: Error message

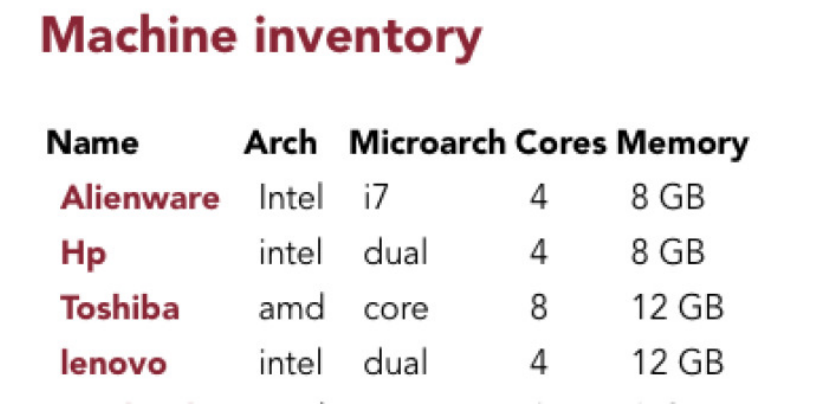

Figure 5.7: Before update

Figure 5.8: Editing machine details

Filtering inventory

This functionality helps the user to filter the inventory using the context of desired information. For example, the user is able to filter the machines inventory with memory size or reservation dates. 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 drop down menu user select a context option to search for. For example, user can filter the machine inventory by selecting a kind of microarchitecture.

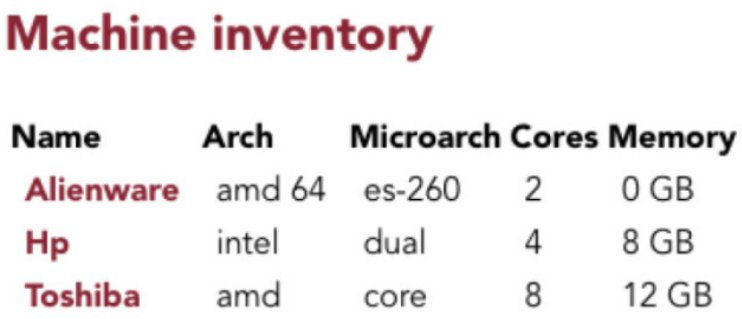

Figure 5.9: Updated machine properties

5.2 Performance Evaluation

We have evaluated the performance of this software by comparing the performance of the functionality with different sizes of data. The reason for this process is to observe how the software will perform as the system grows in number of machines. For each requirement tested we recorded their network request speed and the layout speed. We used random generated data to increase the size of machines from 10 to 1000, and performed this same evaluation accordingly. Figure 5.10 shows the performance of the software where we compared the event of searching for a machine in the system. The graph points progress partially linear as the data increased. Figure 5.11 and Figure 5.12 as well shows same progression, and that means that the performance is steady with a slight change in speed as the system data increases.

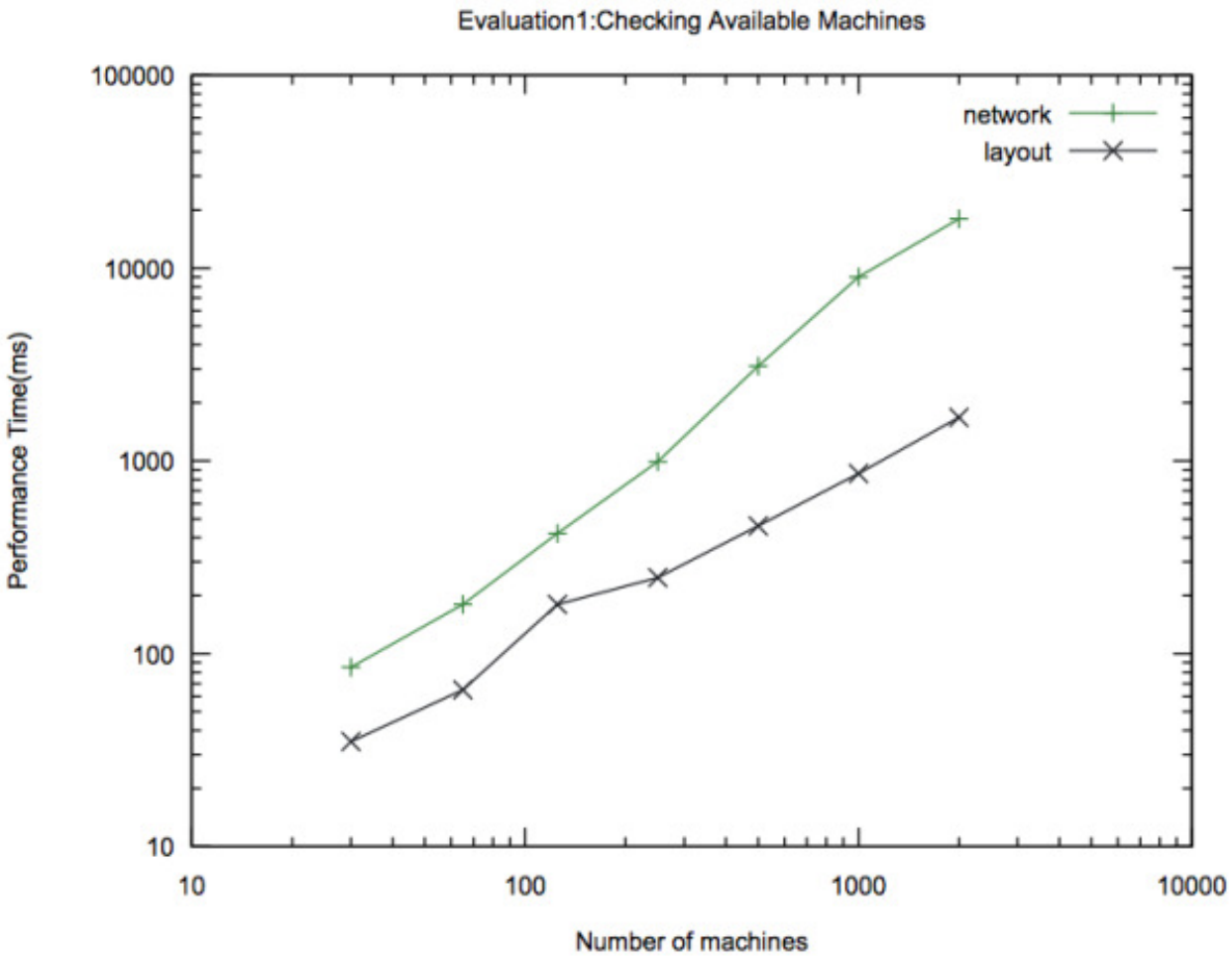

Figure 5.10: Evaluation of software on searching inventory

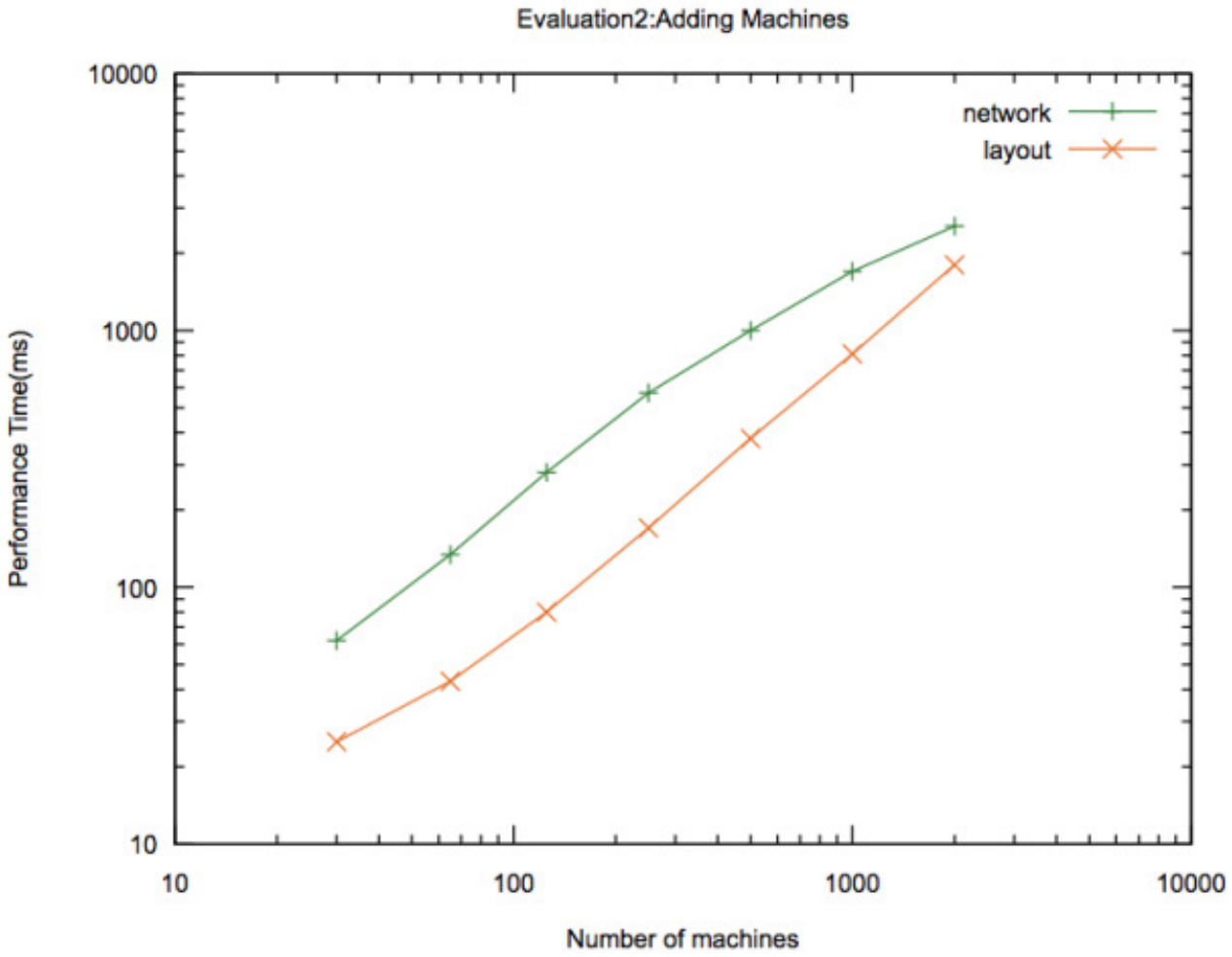

Figure 5.11: Evaluation of software on adding machines

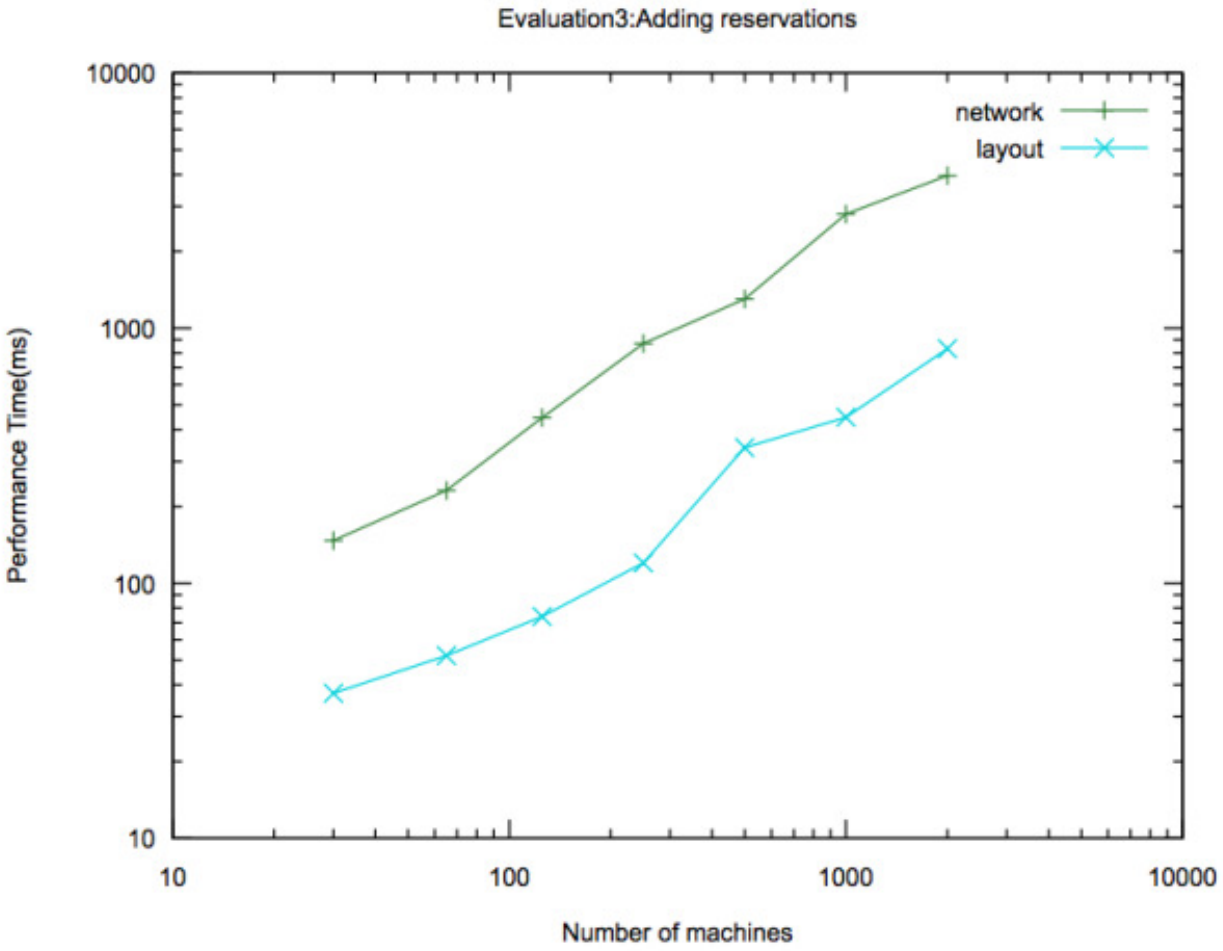

Figure 5.12: Evaluation of software on making reservation

6 Conclusion

The Clowder system, which was designed to manage test machines, has been completed by addressing the issues of system management and accessibility. It was required to have a flexible user interface for dynamic accessibility, and a functional database system for storage and management. These issues have been resolved by designing a front-end and extending the back-end software that provides the required functionality on the user interface and the database. We have extended a web interface where users can communicate with the test machines by sending requests via the HTTP protocol. This web interface contains the required functionality for making reservations, updating machine details and viewing the inventory and has generally resolve the issue of user accessibility. Also, as part of the requirement to complete the Clowder system, we extended a SQL database for storing and retrieving data. This database enables the Clowder system to store and manage data of users activity on the test machines and their information. The database we have created has the ability to plug in a DHCP server between Clowder and the test machines. All these components has accomplished the requirement for this project and now made it easier for researchers to use the Clowder system without much complexity.

Bibliography

[1] A. A. Donovan and B. W. Kernighan. *The Go programming language*. Addison-Wesley Professional, 2015.

[2] S. Gadgil, S. Pillai, and S. Poojary. Sql injection attacks and prevention techniques. *International Journal on Recent and Innovation Trends in Computing and Communication Volume: 1 Issue: 4 293–296, APR 2013*, 2013.

[3] C. Gerald. Dynamic host configuration protocol. Available at: `https://wiki.wireshark.org/DHCP`.

[4] M. Johnston. Dhcp preboot execution environment (pxe) options draft-ietf-dhc-pxe-options-01. Technical report, txt, Internet-Draft, 2005.

[5] J. Kreibich. *Using SQLite*. O'Reilly Media, Inc., 2010.

[6] T. G. P. Language. Go listen and serve, August 2017. Available at:`https://golang.org/pkg/net/http/#ListenAndServe`.

[7] M. H. Pirahesh, T. Y. Leung, G. M. Lohman, E. J. Shekita, and D. E. Simmen. Optimization of sql queries using early-out join transformations of column-bound relational tables, Aug. 20 1996. US Patent 5,548,758.

[8] L. Rosenfeld and P. Morville. *Information architecture for the world wide web*. O'Reilly Media, Inc., 2002.

[9] SQL. Sql joins explained. Available at: `http://www.sql-join.com`.

[10] H. C. Subramanian. Automatic allocation of least loaded boot server to pxe client on a network via dhcp server, Mar. 22 2005. US Patent 6,871,210.

[11] Tutorialspoint. Sqlite clause and usage, 2017. Available at:https://www.tutorialspoint.com/sqlite/sqlite_and_or_clauses.htm.

[12] S. Varghese. Structs and interfaces. In *Go Recipes*, pages 53–74. Springer, 2016.

Appendices

A Appendix A

Listing A.1 shows the basic python script used for random generated machines and reservations.

Listing A.1: Script for generated data

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3
4 import sqlite3 as lite
5 import sys
6 import datetime
7 import random
8 from datetime import date
9
10 con = lite.connect('clowder.db')
11
12 with con:
13
14     cur = con.cursor()
15
16     cur.execute("INSERT INTO Machines WITH RECURSIVE mch(id, name, arch, microarch,
           cores, memory)AS(SELECT random(),abs(random()% ),abs(random()%),abs(random
          ()%),abs(random()%),abs(random() %)UNION ALL SELECT random(),abs(random() %)
          ,abs(random()%),abs(random()%),abs(random()%),abs(random()%)FROM mch LIMIT 0
           ) select * from mch;")
17
18      cur.execute("INSERT INTO Reservations WITH RECURSIVE res(id,user,machine,start
          ,end,ended,pxepath,nfsroot)AS(SELECT random(),(SELECT users.id from Users)
          ,(SELECT machines.id FROM Machines),abs(random() (strftime('%s','2017-10-31
           23:59'))),strftime('%s','2017-01-01 00:00') + abs(random()% (strftime('%s
          ','2018-01-31 23:59') - strftime('%s','2017-01-01 00:00'))),NULL,abs(random
```

```
        ()%),abs(random()%)UNION ALL SELECT random(),(SELECT users.id from Users),(
        SELECT machines.id FROM Machines),abs(random()% (strftime('%s','2017-01-31
        23:59'))),strftime('%s','2017-01-01 00:00') + abs(random()% (strftime('%s
        ','2018-01-31 23:59') - strftime('%s','2017-01-01 00:00'))),NULL,abs(random
        ()%),abs(random()%)FROM res LIMIT 0 )select * from res;")
19 if con:
20     con.close()
```

B Appendix B

Random generated number of machine and reservations for Figure 5.1.

No	Number of Machines	Number of Reservations
1	5	10
2	10	5
3	15	550
4	20	35
5	50	180
6	100	670
7	150	120
8	250	1000
9	500	50
10	1000	180
11	2000	1600

Table B.1: Scattered plot data

The following table shows the data generated from Network and Layout time of the software performance as shown in Figure 5.10, Figure 5.11 and Figure 5.12.

No of Machines	Network Time(ms)	Layout Time(ms)
30	85	35
65	180	65
125	420	180
250	991	248
500	3100	460
1000	9000	860
2000	18000	1680

No of Machines	Network Time(ms)	layout Time(ms)
30	62	25
65	134	43
125	280	80
250	571	170
500	1000	380
1000	1700	810
2000	2550	1800

No of Machines	Network Time(ms)	Layout Time(ms)
30	147	37
65	231	52
125	445	74
250	870	120
500	1300	340
1000	2800	447
2000	3960	830