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

April 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

List of Tables

List of Figures

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:

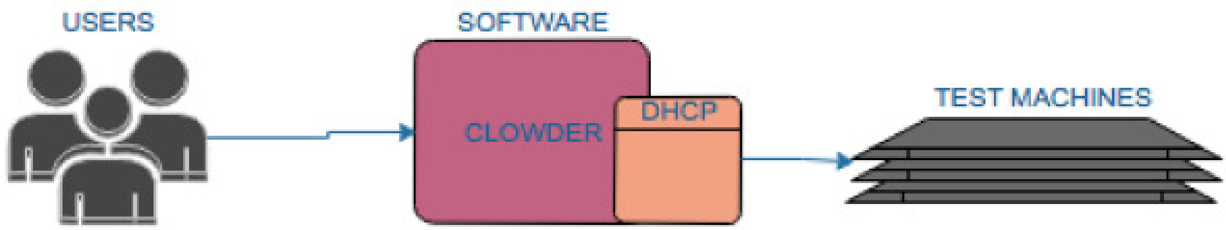

Figure 1.1: Design

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[3]. The operations of standard DHCP servers that serve up IP addresses will not be disrupted by the use of the extended protocol [5]. 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 [5]. 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 a 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. 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). 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, OUTER, LEFT and RIGHT join. This concept was used in this project to accomplished some of the functionality we have specified.

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

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 machine 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.

Searching Inventory

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 reservations. 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.

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.

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) and the control protocols. Figure 3.1 describe the main structure of the design in terms of operation. We designed a web interface that provides flexible

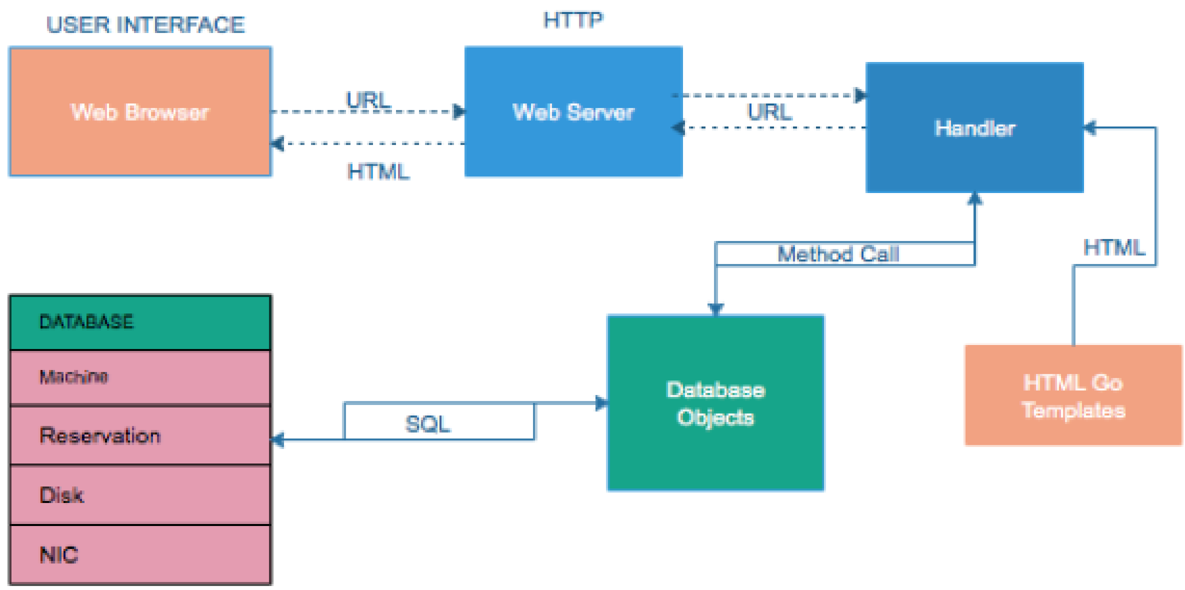

Figure 3.1: The Design

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:). 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 as a channel between the user and the database. We designed a 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 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 HTML post request [6]. 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 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 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 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.

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

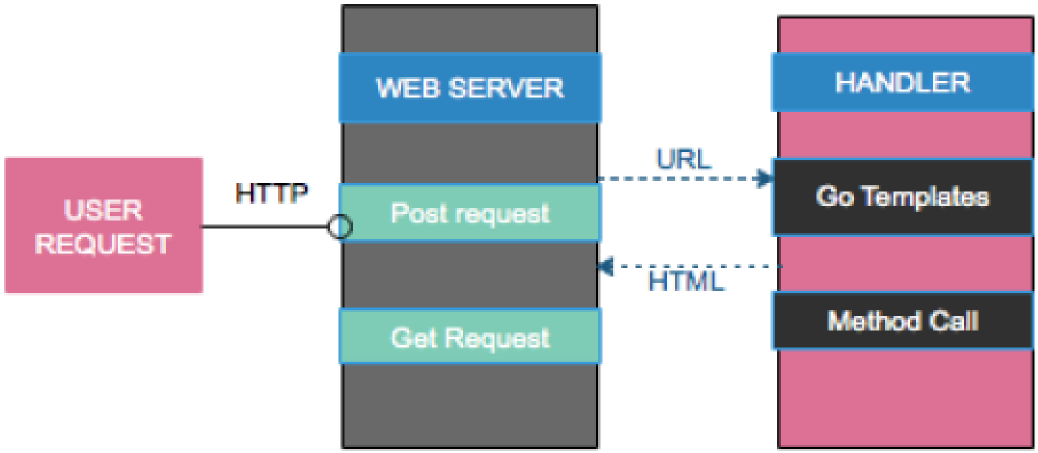

Figure 3.2: Server Activity Description

back end tech. SQLite was used here as prototype because it is fast and easy to set up. 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.

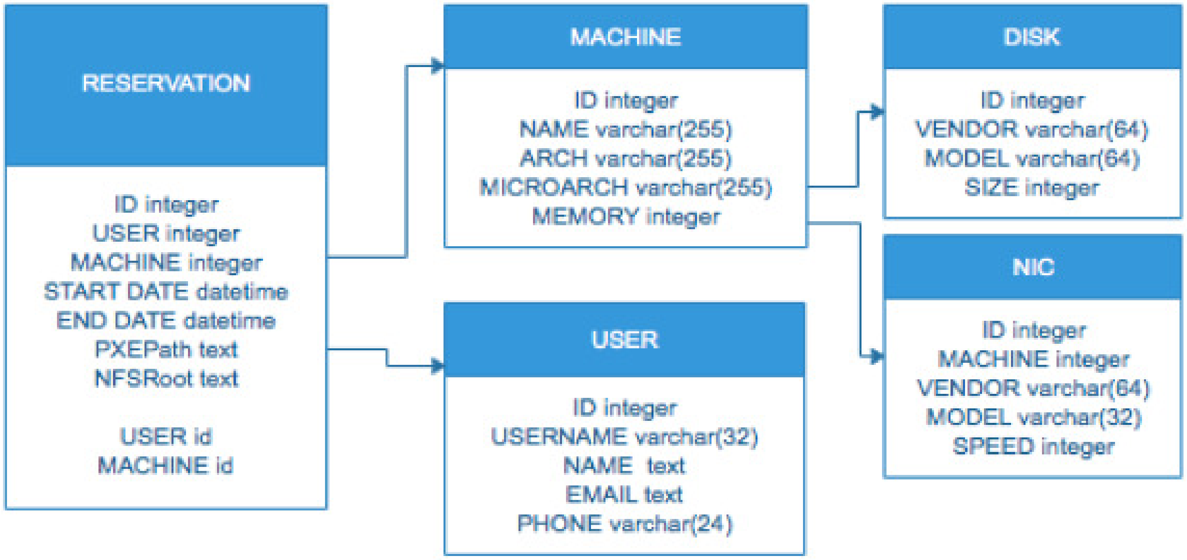

Figure 3.3: Database design

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 received from the HTML form to be stored in the database. The server initiate 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 specific Go function is called to fetch the data from the designated database. This process is the protocol that controls the sequence of data flow in the system. It controls data input, handling and output. This protocol help 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.

4 Implementation

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. This was accomplished by taking measures provide maximum result as required. This measures include the choice of tools used for the development and the design approach. It was implemented to demonstrate the realization of the proposed specifications especially features such as making reservations in the system.

4.2 Implementation tools

The user interface is implemented on the internet web browser using local server port as web address. The web browser serves as the environment for testing and implementing the front-end (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. The Go HTTP server provides connections between the front-end and back-end implementation. It servers as a channel for parsing request through user interface and database. We used a local TCP network for this implementation. 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 development 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 written two main package to actualize the development goal. These packages includes:

- Database package
- HTTP package

The database package contains all the Go files with database related struct (class). 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 struct and database resources. The HTTP package also contain some Go file relates to the user interface (for example the function Handler) and HTML files. the HTML files contains 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 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 packages

4.4 List of Struct and methods

In Go programming language a Struct is same as Class. It contains group of methods that perform the tasks specified in the software requirement in our design. It also methods contains the algorithms that process different queries and schema actions. The method contains fields that represent the entities of the hardware in the system, and this fields are used as argument in the various functions.

No	Struct	Methods
1	machine	initMachines()
		CreateMachine()
		UpdateMachine()
		GetMachines()
		GetAvailableMachine()
2	reservation	initReservation
		CreateReservation()
		GetReservation()
		FilterReservation()
		SortReservation()
3	disks	initDisks
		CreateDisks()
4	nics	initNICs()
		CreateNICs()
5	user	initUsers()
		CreateUser()
		GetUser()

Table 4.2: List of Struct and Methods

4.5 Description of Methods

Adding Machines to the system

Adding machines and other devices to the system is one of the software requirements in section 3.1. 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 InitMachine method create a new database schema with defined table fields where data is stored. When the user submits a Add Machine form, the InitMachine create the database schema where the machine details is stored. It uses type (Machine) fields as argument for creating the schema fields. 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 method is responsible for parsing the data to the database schema. It uses the SQL 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 form data according 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:

PSEUDOCODE:

1 Set the table fields . 2 Insert into values into the fields. 3 If value format is valid, 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 (
                        //field values
                )`,
            name, arch, microarch, cores, memoryGB)
    return err
}
```

Making a Reservation

The reservation process is similar to creating a machine but here it requires the User ID and Machine ID as foreign keys in creating the database schema. 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.

PSEUDOCODE:

1 Input reservation values in the field. 2 Select machine id from the list where machine name is chosen. 3 Select user id from the list of user where name is chosen. 4 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 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.

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 as one of the requirement 3.1. 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 machine, the user enters a specific start date and end date in the provided text feild, and submits the query. After submiting the query, the HTTP handler processes this request, and the server calls the GetAvailableMachine method 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 [4]. The server uses ListenAndServe method to listen to incoming request and to call the requested method to handle the request. In this method, we used a SQLite JOIN sub query 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, OR and AND operator to perform comparisons between reservation dates and time. [8]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[2]. Below code listing shows the function and query for this operation: SPEUDOCODE:

1 Set the function arguments. 2 Query the rows and select machine that is not in reservation. 3 Select machine where its reserved and the search dates is between START and END date. 4 And select machine where START or END date is between the search dates. 5 Assign the selected values to the row and return the values. 6 Return values selected

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 web interface using text box that is attached to every field 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 methods

The Get function is a method 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 Response-Writer. The Get methods uses the SQLite selection query to scan through the database and select the required data. Another usage for this Get method 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 specify the machine clicked used its unique ID. This methods is applicable in other struct (type) such as, Disks, NICs and Reservations.

4.7 Problems Encountered

During the implementations, we encountered problems such as formatting plain text to database data types. For example, some functions can parse a plain text as they are to the database, but in some cases where the values have a different data type (eg. Hardware Mac-address data type), this data cant be store as plain text. So this problem was solved by importing special 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 functionality as specified in the requirement. This was achieved by creating multiple cases that proves the performance of the software. This evaluation was performed in real time and directly online. We have included images and table showing the test cases and results.

Checking Available Machines

In the software specification section 3.1, 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 in section 3.1 we have implemented in this software. This is the process where users choose and reserve a machine for definite time and date. When a reservation is made, the details of the reservation appears under the machine being reserved. Figure **??** shows the evaluating process of this function, where we put details of reservation, by selecting a user name, machine and inserting start and end time/date. After submitting with the reserve button, the it appears on the inventory with other reservations list. During the testing we tried to input a wrong date and time format and we got an error as shown in figure 5.3. This proved that the software will not take an invalid date format for any reservation process in the system, rather it will issue a warning to the users appropriately.

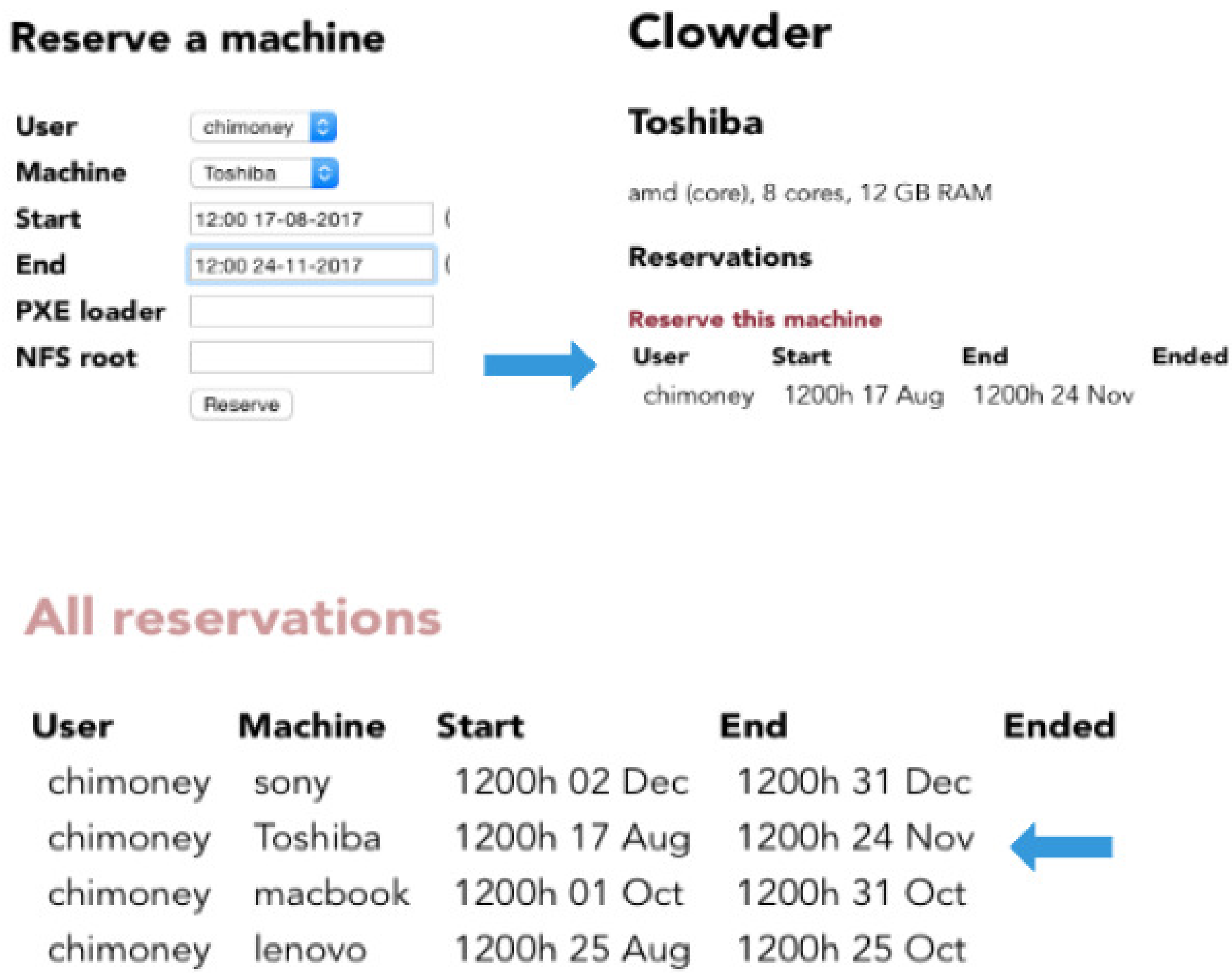

Figure 5.1: Reservation function

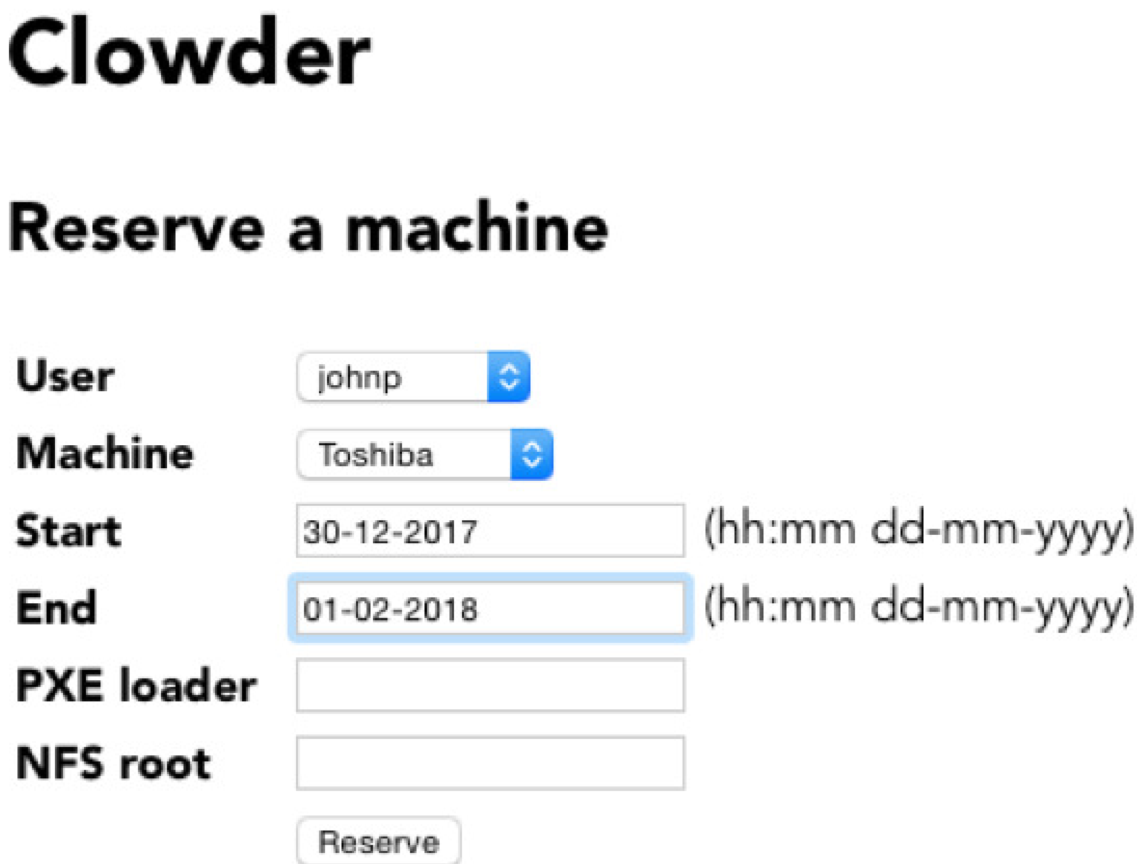

Figure 5.2: reserving with wrong date

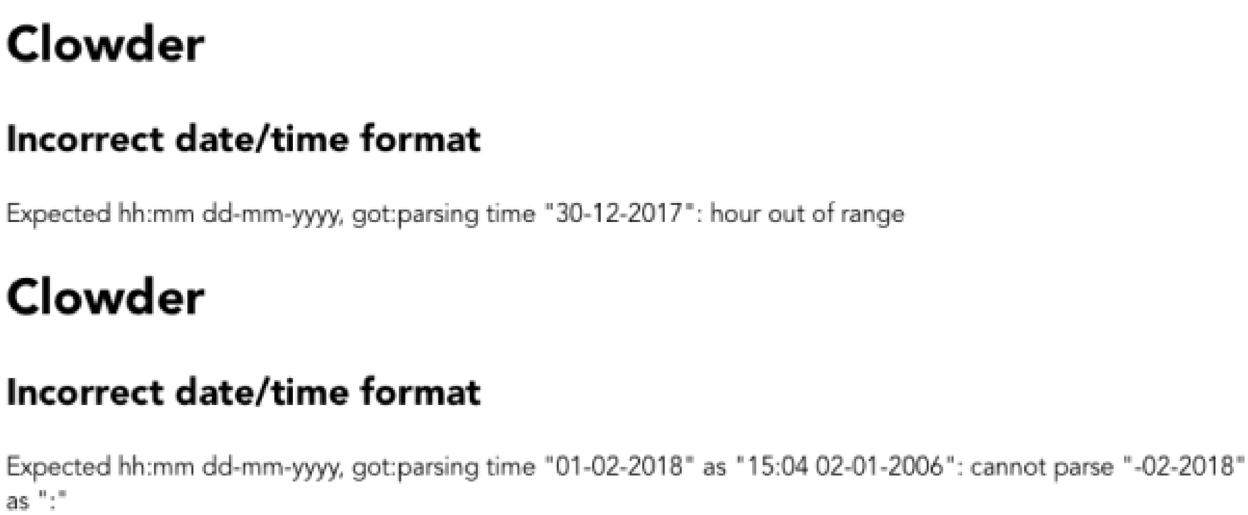

Figure 5.3: Error message

Updating a machine

Update functionality as in section 3.1 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 the process of updating the inventory. This process has proved that this function performed as required because data was collected accordingly and stored with out any error.

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 5.4: Before update

Figure 5.5: Editing machine details

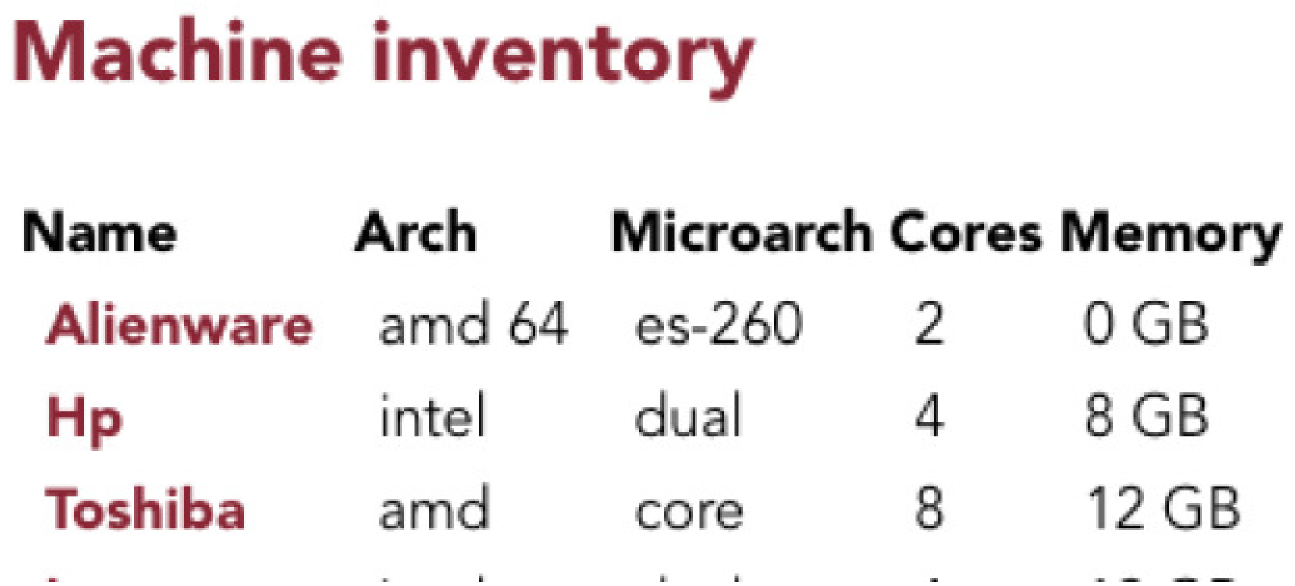

Figure 5.6: Updated machine properties

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 drop down menu user select a context option to search for. For example, user can filter the machine inventory by selecting type OS micro-architecture and architect.

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 5.7: Reservation function

5.2 Performance Evaluations

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 grow in size of machines and other activities. 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 5.8 shows the performance of the software where we compared the event of searching for a machine in the system. The histogram progress linearly as the data increased. Other figures as well shows same progression, and that means that the performance is steady with a slight change in speed as the system data increases.

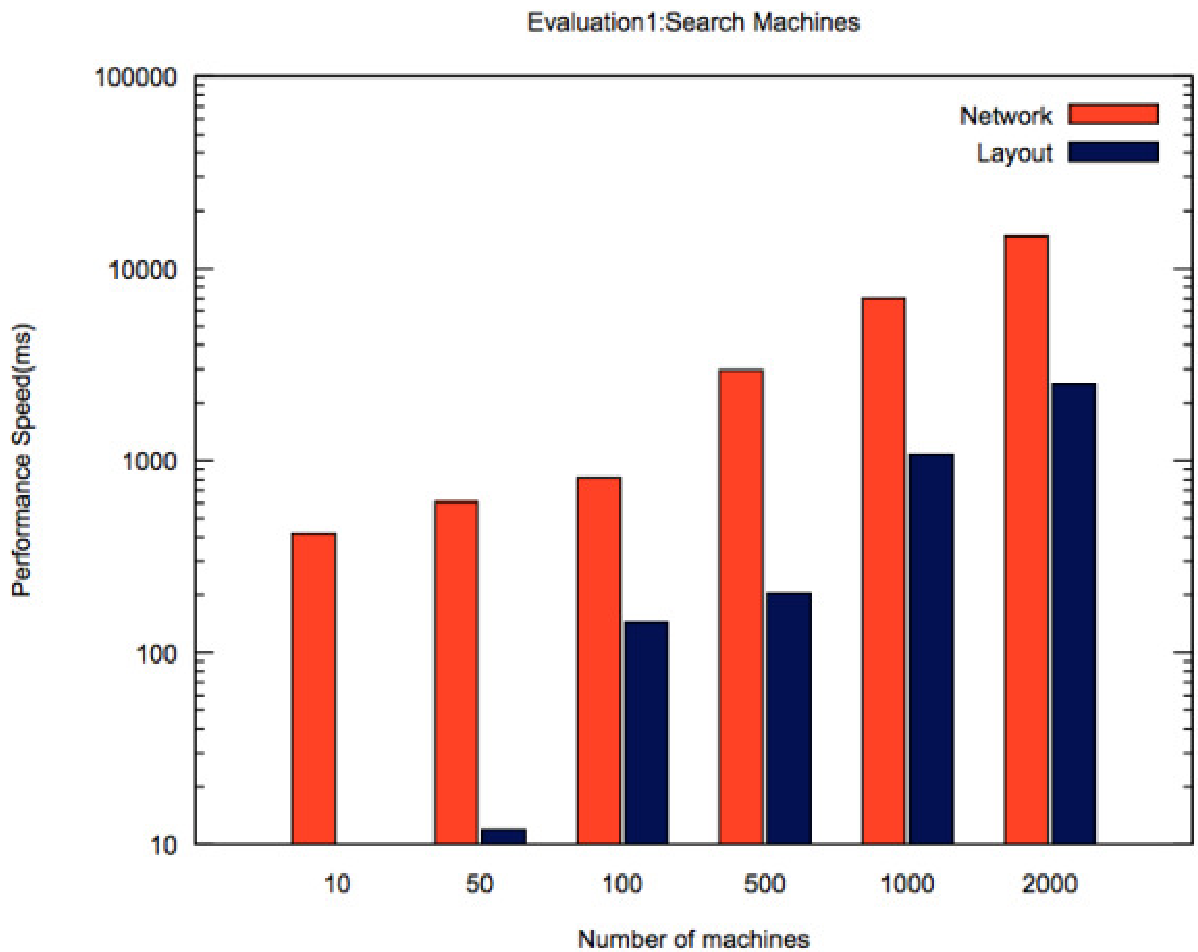

Figure 5.8: Evaluation of software on searching inventory

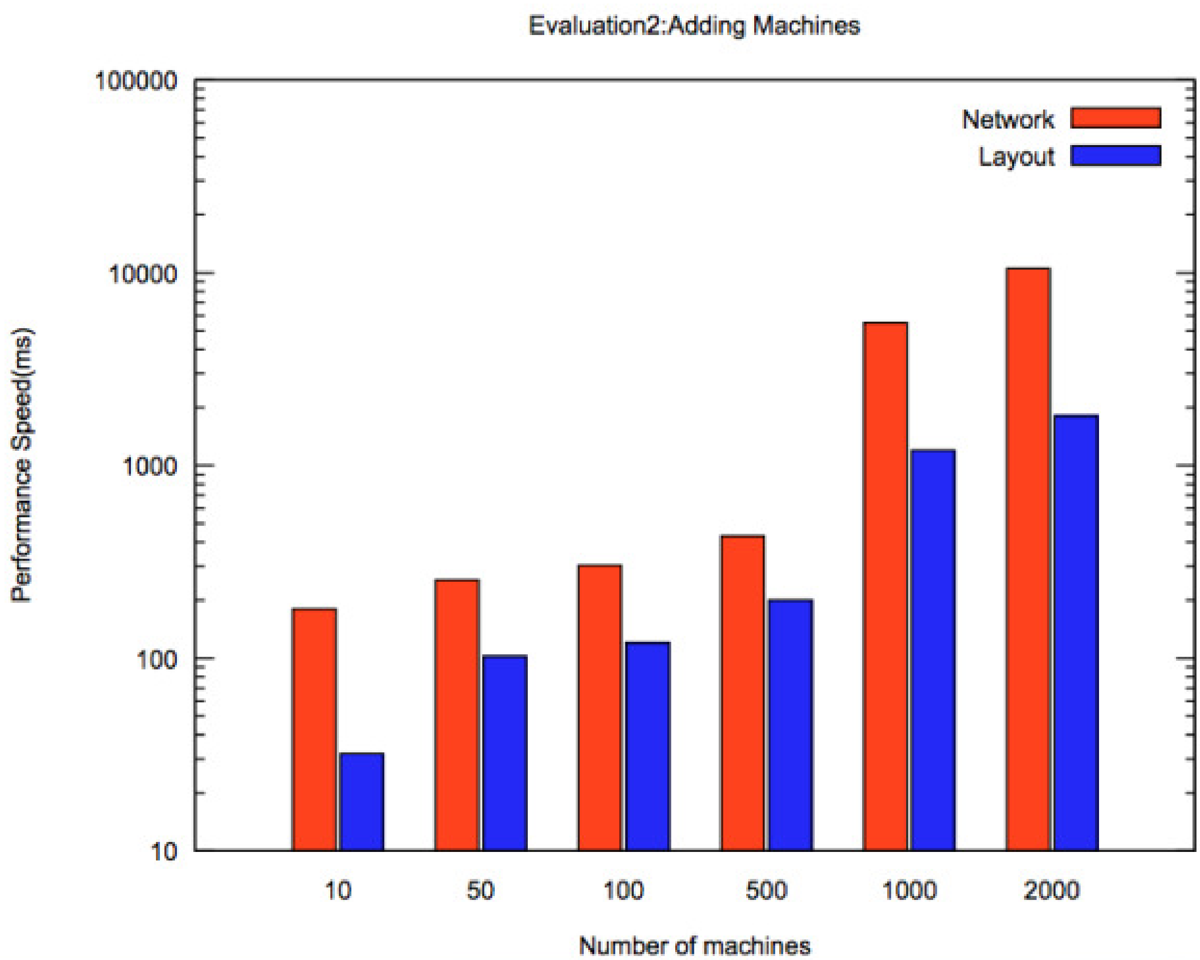

Figure 5.9: Evaluation of software on adding machines

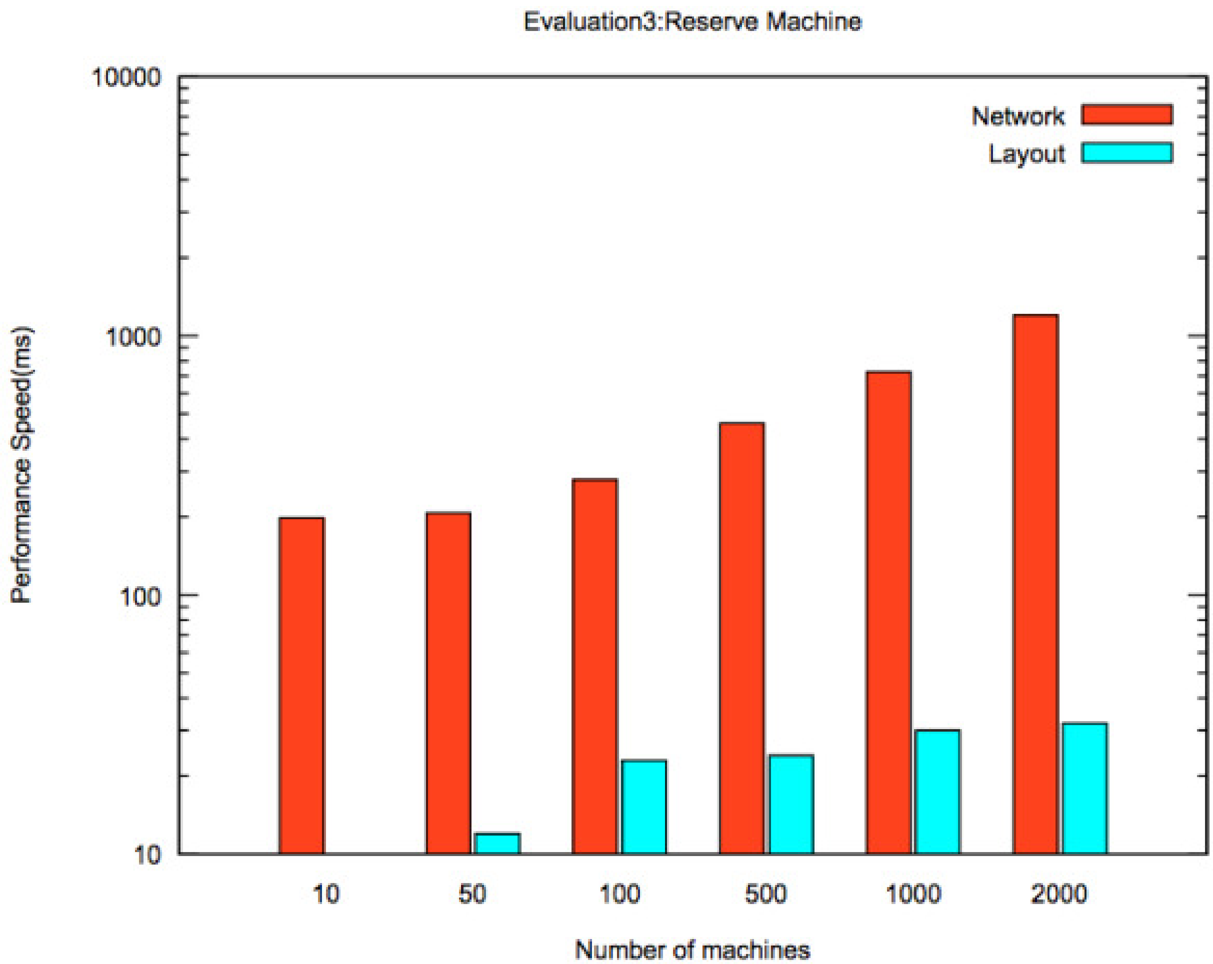

Figure 5.10: Evaluation of software on making reservation

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 management 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 front-end and extended back-end software that provides the required functionality on the user interface and the database. We have created a web interface where users can communicate with the test machines by sending request via Go 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 created 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 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 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] C.C. Network installation using pxe.

[2] S. W. Clause.

[3] GeraldCombs. Dynamic host configuration protocol.

[4] G. HTTP. Go listen and serve.

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

[6] MultiMedia.

[7] SQL. Sql joins explained.

[8] SQLite and O. Clasue. Sqlite clause and usage.