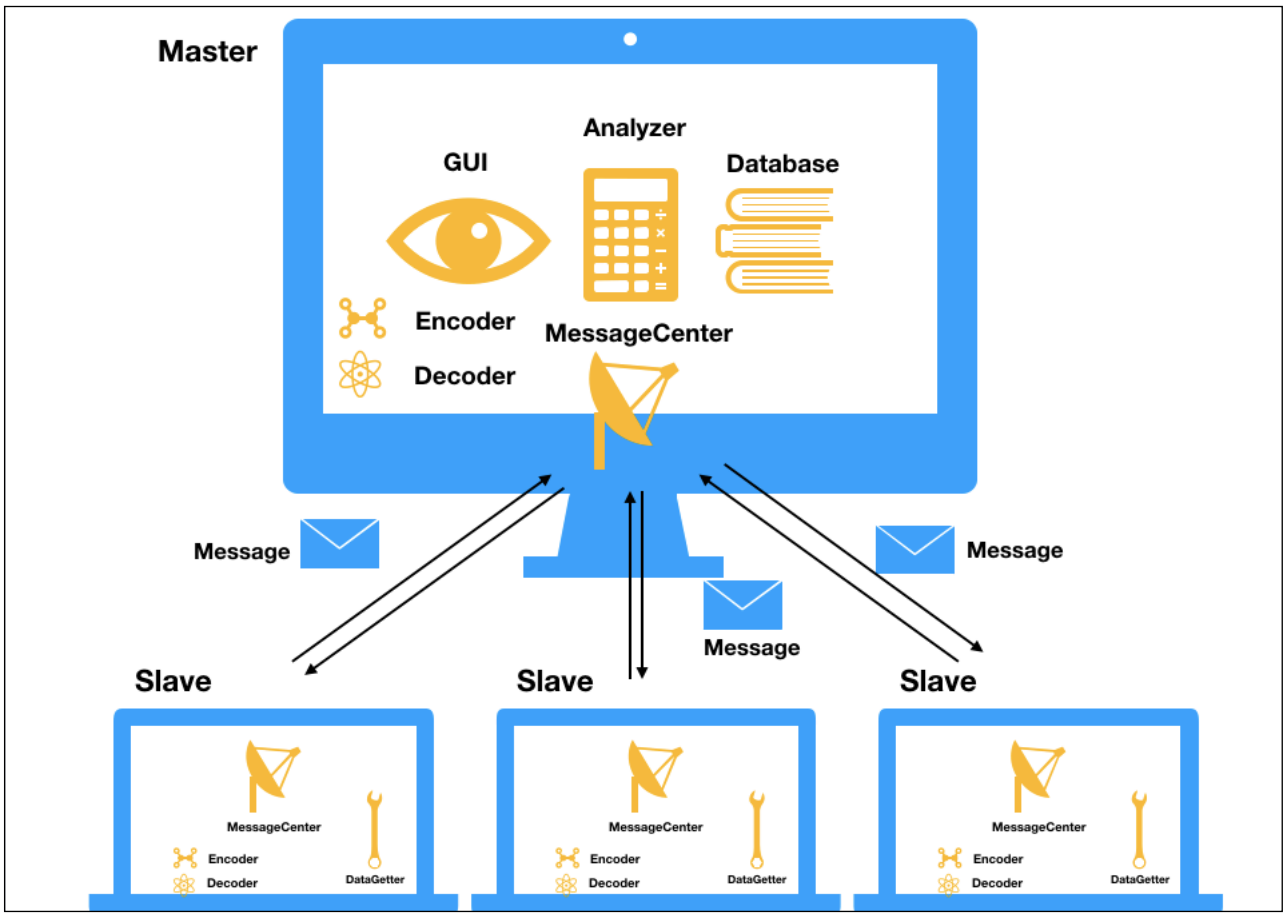
Distributed Resource Monitoring System

C.C. Lee, H.W. Lin, S.S. Jiang - 30 November 2017
Advisor: Asst. Prof. Shih-Wen Ke



Content

Part 1 Introduction

1.1 Overview

We are well aware of the difficulty of building a Distributed-Computing platform. As an alternative approach, it is whether there is a Distributed-Computing-like system that we have the ability to finish in a limited time that should be taken into account by us. As a result, this system, Distributed Resource Monitoring System is born. **This report is divided into five sections which are introduced in the following paragraphs.**

We spend lots of time on surveying a wide variety of background knowledge related to distributed system not only on theories, design patterns for Distributed Computing, but on the applications as well. We study Anaconda, a cluster management tool, for instance. Besides it, the technology of Distributed-Computing is well known by us. These details are elaborated in the second section of this report, Part 2 Survey.

Due to the background knowledge related to distributed systems, we are able to develop our architecture of this system. Besides the survey, we combine the knowledge that we learn in school in this system, such as the technique of Objected-Oriented Programing and SQL database. The details of Distributed Resource Monitoring System is introduced in the third section of this report, Part 3 Distributed Resource Monitoring System.

Nothing is perfect. We acknowledge the potentiality of this system as well as the defects of this system. We do a brief aftermath related to the feasible developments in the future, which are elaborated in the fourth section of this report, Part 4 Future Work.

Lastly, the report ends with a summary, Part 5 Summary, to conclude our result as well as some points that are worthy to be noticed.

1.2 Objectives

Managing a data center

It is known to all that managing a data center is a heavy task. There are hundreds even thousands of computers that need to be monitored and maintained. This system should be an aid for those who have the responsibility to keep computers running regardless of the circumstances.

Simple interface

The main function of this system is to provide a simple interface to users so as to monitor the status of machines. Users can see various kinds of information through the GUI, Graphic User Interface. Basically, the GUI provides the hardware information, the OS, Operating System, information, and the processes information. In addition, the update frequency of the information is changeable.

Managing processes through the GUI

Monitoring is not enough. We provide a function that users are able to kill a process listed on the GUI, even though there is an alternative approach to kill a process; thus, users control the slave machine remotely by using ssh protocol. Instead of the ssh protocol, users do not need to log in on the slave machine since the one that undertakes the task is the application running on the slave machine. The account that kills the process selected by the user shall be as same as the one that starts the slave application. However, the condition to make this function available is to make sure that the account has the permissions to perform such action.

Frontend and Backend friendly

There are also a number of minor functions. This system adapts logging and sqlite. With the logging function, users can trace the history of the system status. With the sqlite, users can perform a wide variety of tasks. Also, debug mode is available for users. The applications automatically read the configuring file to determine whether to execute in the debug mode or not. Those functions will be elaborated in more detail in the following report.

1.3 User Requirement Definition

In this project, we can briefly classify functions into a few categories.

Monitoring

Monitoring slaves is the main function of this project. However, there is a tremendous amount of information that describes the status of a computer. Alternatively, we focus on three fields which are hardware, operating system status and process. For hardware, we record the clock frequency, the numbers of the physical cores, the number of logical cores and the size of Memory. We believe that these data are detailed enough to be aware of the status of slave machines. For operating system status, we record the maximum, minimum and present value of CPU and Memory usage. This data is critical for the function of Alarm, which is introduced in the following text. For processes, we record the CPU and Memory usage of each process so that we can easily notice what process occupying the resource and take actions under that circumstance.

For Monitor, we record:

1. Hardware status: Logical core, Physical core, Minimum CPU frequency, Maximum CPU frequency, Memory size.
2. OS status: CPU user usage, CPU system usage, Idle CPU, Memory usage.
3. Process status: List all processes which use the system resource.

Command

Except monitoring, we can command slaves as well. Making command means that we can remotely access our slave application of the host instead of using Ubuntu terminal. The commands include stop slave, start slave, terminal slave, set updating frequency and kill process. Starting transmitting the data from a slave can be done by“start slave” command, and it can be stopped by either “stop”or “terminal” command. In addition, we can decide the transmitting frequency by “set updating frequency” command. Once we figure the process which make CPU or Memory overloading, the “kill process” command can help us terminate the process.

For Command, we can use:

1. Slave command: Set alarm, set updating frequency, send stop message, send start message, send terminate message.
2. Process command: kill a process.

CML, Command Line Interface

CML have to be used for the very first time executing this system to set the machines information on both master side and slave sides. After that, users can still use CML to manage all machines settings; in addition, we can turn off debug mode by editing debugsetting.cfg file to hide the CML, to skip the machine setting step, and use previous settings to execute.

GUI, Graphic User Interface(Figure 1-2)

On the GUI, users are able to view the followings,

1. The names of slaves that are monitored.
2. The hardware status of each slave, such as physical cores, logical cores, the size of Memory.
3. The status of OS running on monitored slaves, such as current CPU usage.
4. The processes running on slave that its CPU usage is not zero.
5. The update rate of each slave machine.
6. The alarm threshold of each slave machine.

Through the GUI, users are able to perform the following actions,

1. Terminating the master application through the “Master” tool bar.
2. Refreshing the slave hardware information through the “Master” tool bar.
3. Changing the refresh rate of the display table through the “View”tool bar.
4. Changing the update rate of each slave machine, separately, by right-click the machine name.
5. Asking the slave machine to start transmitting, separately, by right-click the machine name.
6. Asking the slave machine to stop transmitting, separately, by right-click the machine name.
7. Asking the slave machine to terminate, separately, by right-click the machine name.
8. Setting the alarm thresholds of each slave machine, separately, by right-click the machine name.
9. killing the selected process listed on the display table.

Alarming

As a monitoring system, we did not expect administer can sit in front of desk all day long. Hence, we get an automatic alarm mechanism. In other words, the system will pop out the alarm window and sending email to the administer while CPU or memory was overloading, such that the administer can handle the problem at first hand. (Figure 1-1)

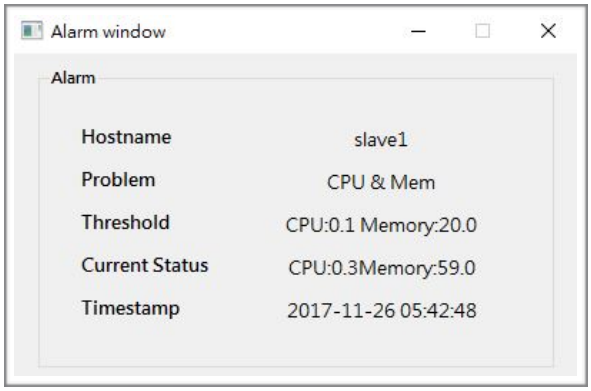


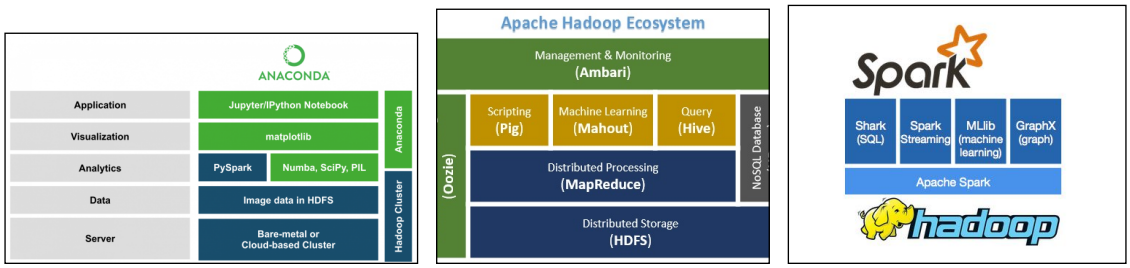
Figure 1-1

Part 2 Survey

2.1 Overview

It is known to all that to have a balance between theories and practices is more than tough. In the last semester, we make lots of attempts in a number of fields to broaden our knowledge related to Distributed-Computing. However, what we have done are only theories. What we focus on in the past four months is how to utilize these knowledges into practice.

There is no denying that Distributed-Computing is one of the most robust technologies not only now but also in the future. Nowadays, there exists a wide variety of applications of Distributed-Computing, Data Science for instance. With the aid of Distributed-Computing, the performance of Data Analysis can be enhanced. Machine Learning is another example. By using Apache Spark, a Distributed-Computing platform, the engineers can achieve what they want with more ease.



Anaconda, Apache Hadoop, Apache Spark

We keep trying to understand how Distributed-Computing works. What sorts of tasks should the machine, served as a master, do? How does it manage tasks? How does it insure the CIA, the confidentiality, the integrity, and the availability, of the data? How do the slaves communicate not only with each other but also with the master? There are a considerable number of problems that deserve our attention. In a limited time, it is unlikely for us to understand all of them. As a result, we want to focus on the synchronization of the data. At first, we start from theories, such as the design patterns for Distributed-Computing. Following by theories, we begin to experience the applications of Distributed-Computing.

In the following of this section, we elaborate our survey proceed before this system is developed. They are crucial to this project. Design patterns for Distributed-Computing brings us problems related to Distributed-Computing. Anaconda Cloud gives us a concept of a cluster managing tool while Apache Spark offer us a brief understanding of Distributed-Computing firmware. HDFS provides an approach to manage the data in a distributed way. With the experience mentioned above, this project becomes possible.

2.2 Design Patterns for Distributed-Computing

Design patterns for Distributed-Computing brings us problems related to Distributed-Computing.

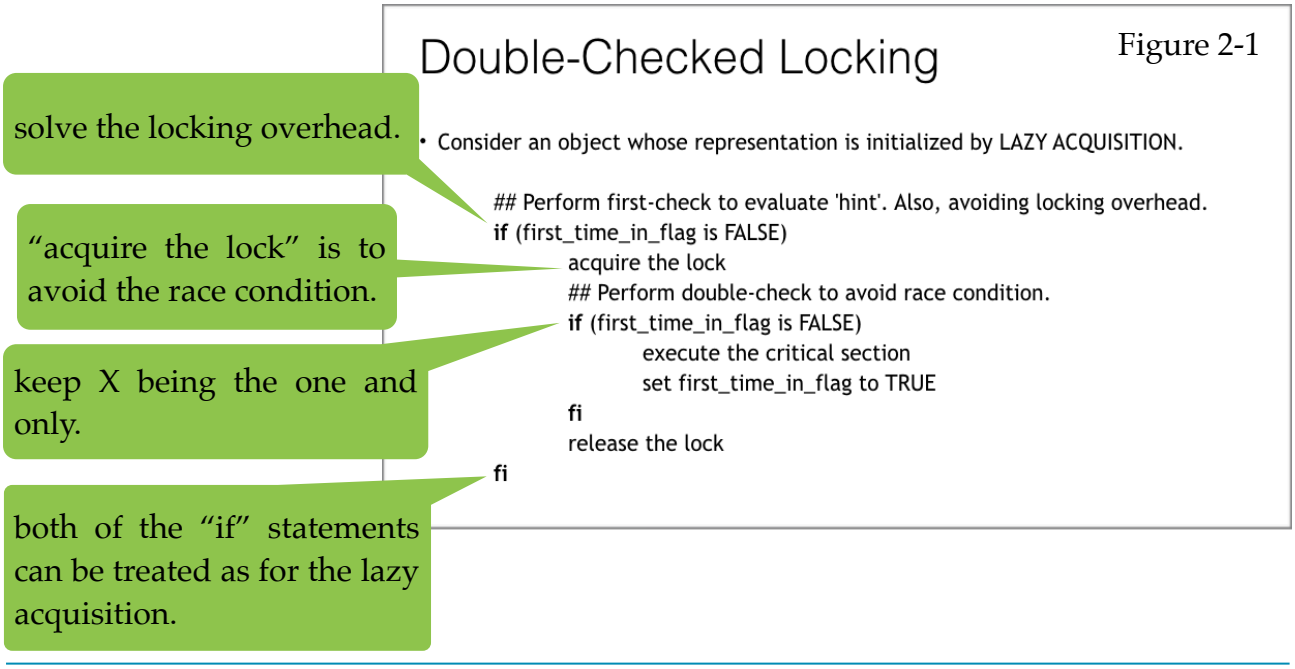
We adapt a reference book, Pattern-Oriented Software Architecture written by Frank Buschmann, Kevlin Henney, and Douglas C. Schmidt, as one of main references. This book leads us to this field as a pioneer. In this book, it indicates a number of problem domains that Distributed-Computing may encounter. This book mentions a dozen of design patterns that target on the problem of the Synchronization. In the following text, **we design a scenario to examine one of the design patterns that we acknowledge.**

Double-Checked Locking, a design pattern to solve problems of share resource.
(Figure 2-1)

1. We have three nodes, A, B, and C, and a server, S.
2. After submitting the task to every node, A, being the first one, request an Object, X.
3. X is a **lazy acquisition** object, and it should be stored in S.
4. Since A is the first node who requests X, obviously, X is not implemented yet.
5. There is a situation that when A is implementing X, simultaneously, B and C also request X. Therefore, a race condition is one of our concerns.
6. Also, we have to keep X is the one and only.
7. In addition, the problem of the locking overhead is taken into account.

Let's do a summary. The goals are as followings.

1. Lazy Acquisition - To initialize the object X as lately as possible and to make sure the critical section be executed for only once. As a result, the object X would be the one and only.
2. Prevent Locking Overhead - To avoid the unnecessary lock acquirement.
3. Avoid Race Condition - Making sure the critical section is executed by only one process at a time.



2.3 Cluster Management Tool- Anaconda Cloud

Anaconda Cloud gives us a concept of cluster management tool

We experience the functions of Anaconda Cloud, which at first, it is one of the steps that to configure a Distributed-Computing platform, such as Apache Spark and Apache Hadoop. Anaconda Cloud is a management tool used in a cluster system. It provides several useful approaches to manage a cluster, for instance, the user is able to install an application on every computer at once by operating on master machine through the Anaconda Cloud interface. We stimulate our needs on a virtual machine environment. The virtual machines are based on Ubuntu OS since we believe that a linux-based OS can provide us a flexible circumstance.

ssh Protocol

We spend some time on configuring ssh protocol. We realize that the ssh protocol is the essential part of Anaconda Cloud. Since we are not familiar with the Linux-Based System, we learn how it works, how to use it, …etc.

RSA

RSA plays an essential part of the processes of configuring Anaconda Cloud. Given that Anaconda Cloud requires the master can connect each slave with ssh protocol without entering the password, RSA is crucial to achieve it. Basically, we have to understand how RSA works in a Linux-Based system and the relationship between ssh protocol and RSA.

2.4 Distributed Storage Filesystem- HDFS

HDFS offers us an approach to manage the data in a distributed way

Speaking of Apache Spark, we cannot not to mention Apache Hadoop. In our understanding, Apache Hadoop provides not only a platform to perform Distributed-Computing by applying MapReduce algorithm but also offers an approach, the Hadoop Distributed File System(HDFS), to manage data in a distributed means. Instead of Apache Spark, which is elaborated later, Apache Hadoop uses the concept of containers to build its architecture.

Since we decide to use Apache Spark as the Distributed-Computing platform, we adapt its filesystem, HDFS. We plan to build our Apache Spark on HDFS.

Why HDSF?

However, configuring HDFS is as tough as configuring Anaconda Cloud, which costs us some time to achieve it. HDFS is a sound way to storage and maintain our data. We realize that transmitting data in a cluster is the most costly work in a Distributed-System. We believe that with the aid of the HDFS, the performance can be increased.

2.5 Distributed-Computing Platform- Apache Spark

Apache Spark provides a brief understanding of distributed computing firmware

After making Anaconda Cloud and the HDFS available in our environment, Apache Spark is the next thing to do. Apache Spark is a well-known Distributed-Computing firmware. It supports several programing language, such as Python. Python is a cross-platform programing language, which is one of reasons that we choose this language to develop our system. To configure Apache Spark is not a easy work since there are tons of configurations.

PySpark

PySpark is the main package in Python that Apache Spark needs to perform Distributed-Computing. Although PySpark package can be installed with ease, to initialize the key components of PySpark cost us a tremendous amount of time.

JuPyter, a IDE for Python

Typically, the user can use PySpark shell to perform Distributed-Computing. However, it is not convenient or user-friendly enough to develop a project with thousands lines of codes. JuPyter is an sound approach. JuPyter is an IDE based on web-brower. It supports several sorts of programing languages, such as Python2 and Python3. Users are able to configure which interpreter that he/she would like to go for. We meet some challenges when we try to get familiar to PySpark library as well as to produce our custom Python3 scripts. Over all, JuPyter is one of the robust Distributed-Computing IDE.

2.6 Task Manager

The monitor interface of the Task Manager.

When we are having troubles with building up an interface, the task manger of Windows system comes into our mind. The task manager of Windows system is exactly what we expect to have. At the meantime, we are aware of the fact that that task manager is too robust for our system; thus, it is a little too complicated. As a result, we adapts a few things of it to our GUI.

AWS, Amazon Web Services

In addition, we also visit the user interface of AWS. AWS is famous for its convenience and reliability of cloud services. We have noticed that the monitoring interface of AWS does not provide the details information of the status of OS, such as the information of processes. This is also a reason that we plan to provide such information in our system.

Part 3 Distributed Resource Monitoring System

3.1 Overview

This section describes this system as detailed as possible. It starts with System Architecture. This part provide not only a big picture of the system but the architectures of two applications as well. Furthermore, we briefly state our development and testing environment. What programing language do we apply? What resources, the API, do we adapt? The last but not the least, we provide a demonstration. Lastly, a manual is listed.

We are willing to apply the principle of Software Engineering in this project given that this project is a rare chance for its characteristics, a four-month project and a group. In the development process, we decide to design this system with an incremental develop model and implement it in **a Object-Oriented approach, referred to 3.2 System Architecture.**

Incremental Development

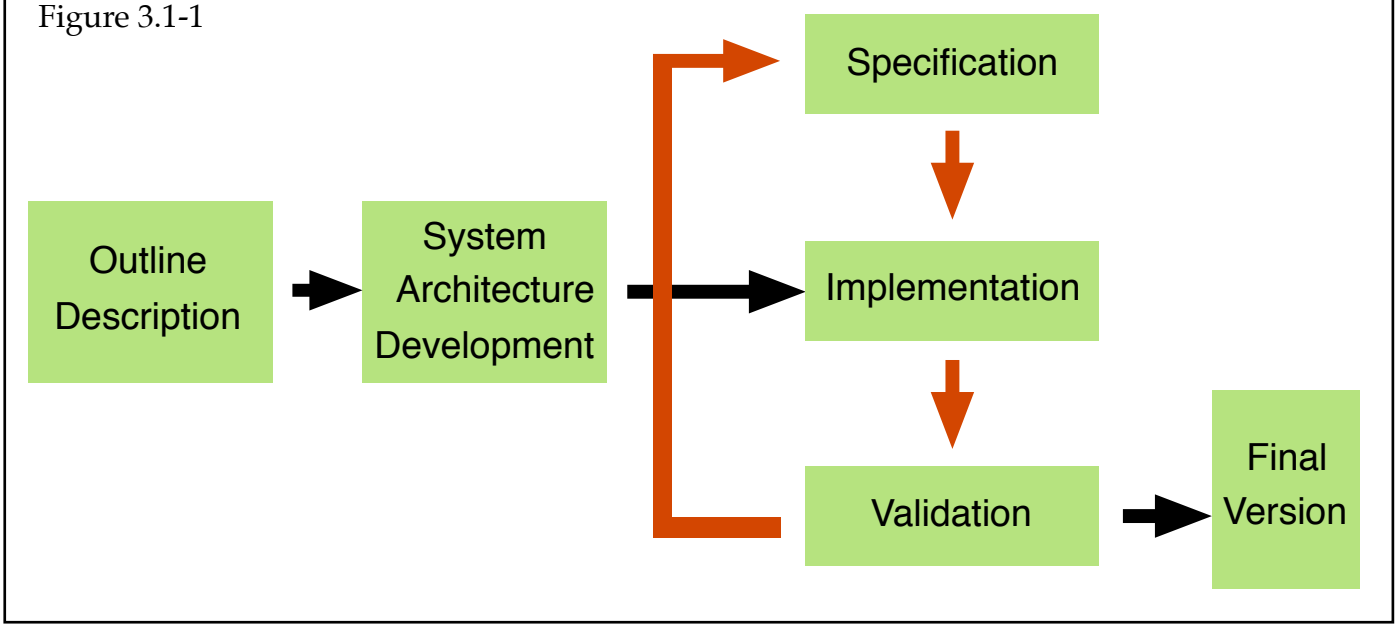
It is known to all that every software development has to go through the following steps:

1. Analysis
2. Requirement gathering
3. System design
4. Development
5. Integration and Testing
6. Acceptance, Installation, Deployment
7. Maintenance

We consider two kinds of development model, the incremental one and the prototyping one. However, due to the limited time and the size of the team, we opt the incremental model.

Basically, Figure 3.1-1 illustrates how we go through the development of this system.

Figure 3.1-1



3.2 System Architecture

3.2.1 Overview

The system combines with a "master" application and a number of "slave" applications(Figure 3.2-1). "slave" and "master" can communicate with each other. By default, "slave" automatically, continuously, send its status to "master". As a result, users can monitor them on "master" easily and perform certain actions. The following is a brief contents of the section 3.2.

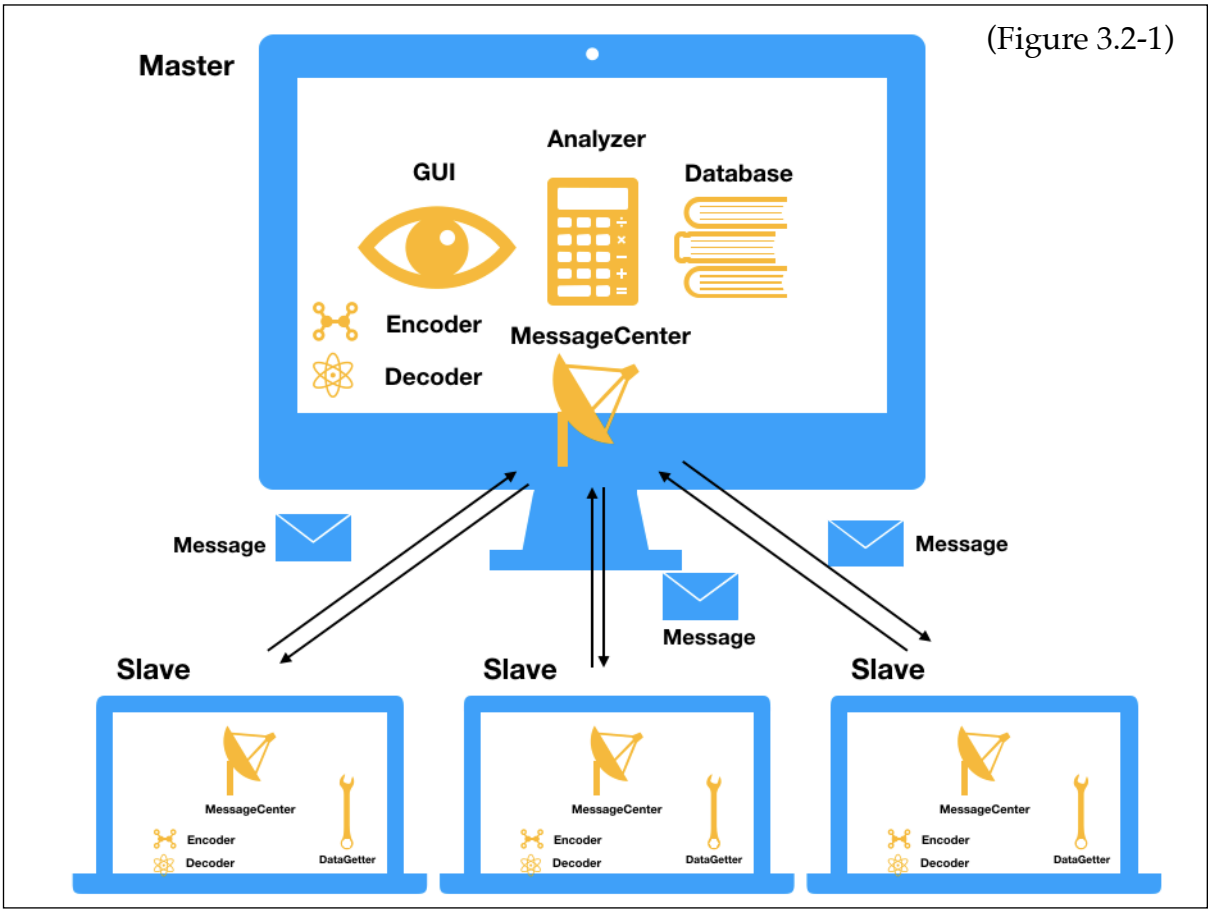
3.2.2 Master Application

3.2.3 Slave Application

3.2.4 The Relationship between Encoder and MessageCenter

3.2.5 The Relationship between Decoder and the receivers

3.2.6 Analyzer



(Figure 3.2-1)

Both "master" and "slave" applications are Python3 script. These applications are assembled by a few reusable components, which be introduced in the following.

1. **MessageCenter**: This object deals with the communication between "slave" and master by using TCP protocol, package “socket” in python3. It sends messages to the target and receives messages from the sender. Also, it executes as a thread.
2. **Encoder**: This object deals with the process of packaging the message from the sender. Similar to the Factory Pattern, a design pattern that is introduced by GoF, according to the system input, the encoder produces a suitable message including every information that the MessageCenter requires, such as the destination.
3. **Decoder**: Instead of the Encoder, this object deals with specifying the content of messages. According to the content, the system has to perform various actions, such as changing the fresh rate as well as changing the type of requesting data.

We also create a number of objects to adapt our requirements of data structures.

1. **Class Machine**, (Figure 3.2-2), stores the information of a machine.
2. **Class Message**, (Figure 3.2-3), is used to communicate with other machine.
3. **Class Status**, (Figure 3.2-4), is the super class of the next three objects.
4. **Class Status_Hardware**, (Figure 3.2-5), stores the data of the hardware.
5. **Class Status_Software**, (Figure 3.2-6), store the data of the software.
6. **Class Status_Processes**, (Figure 3.2-7), store the data of the processes.
7. **Class SystemCall**, (Figure 3.2-8),
8. **Class AlarmData**, (Figure 3.2-9)

(Figure 3.2-2)
Class Machine

name:String
ip:String
port:String

(Figure 3.2-3)
Class Message
destination:String
port:string
content:string
sender:string
createdTime:String

(Figure 3.2-4)
Class Status
machineName:String
type:String
date:String
time:String
fields:String[]

(Figure 3.2-8)
Class SystemCall

sender:String
type:String
content:String

(Figure 3.2-5)
Class Status_Hardware inherits Status

// Methods differ from the parent

(Figure 3.2-6)
Class Status_Software inherits Status

// Methods differ from the parent

(Figure 3.2-7)
Class Status_Processes inherits from Status

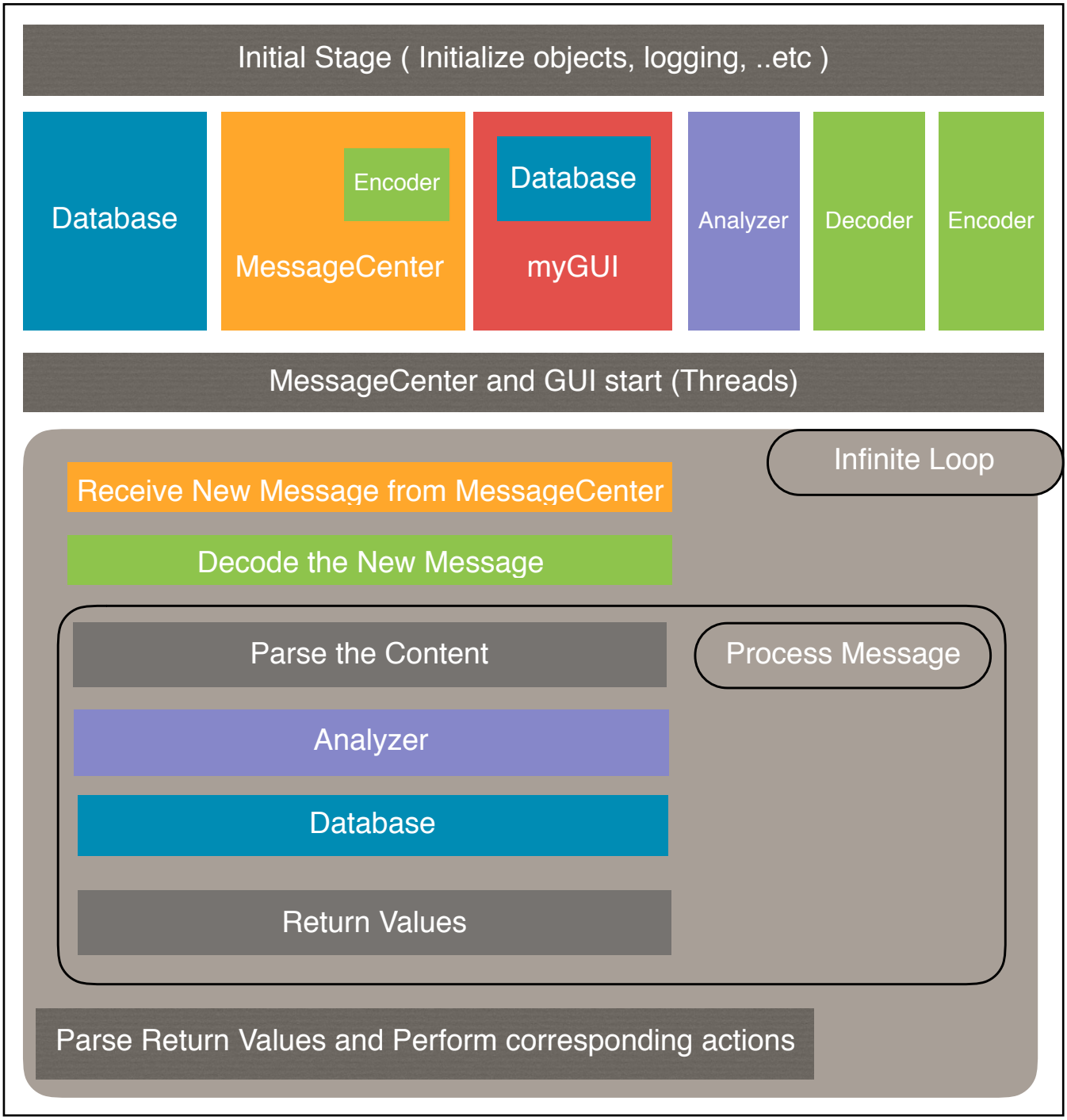
// Methods differ from the parent

3.2.2 Master application

Besides the components mentioned above, "master" adapts three more components describing as the following.

1. **Database**: This object deals with a sqlite database, which stores several kinds of data, such as the messages send/received records as well as the status of slave machines records. It is the only interface between the application and the database.
2. **Analyzer**: This object deals with the analysis of data. For example, it analyze the data to determine whether updates the history record of the memory usage.
3. **myGUI**: This object deals with the user interface. It receives the user request and responds. It executes as a thread.

(Figure 3.2-9 shows the structure of main script of "master")



As the figure 3.2-9 illustrates the architecture of “master”, there are a few crucial points listed in the following.

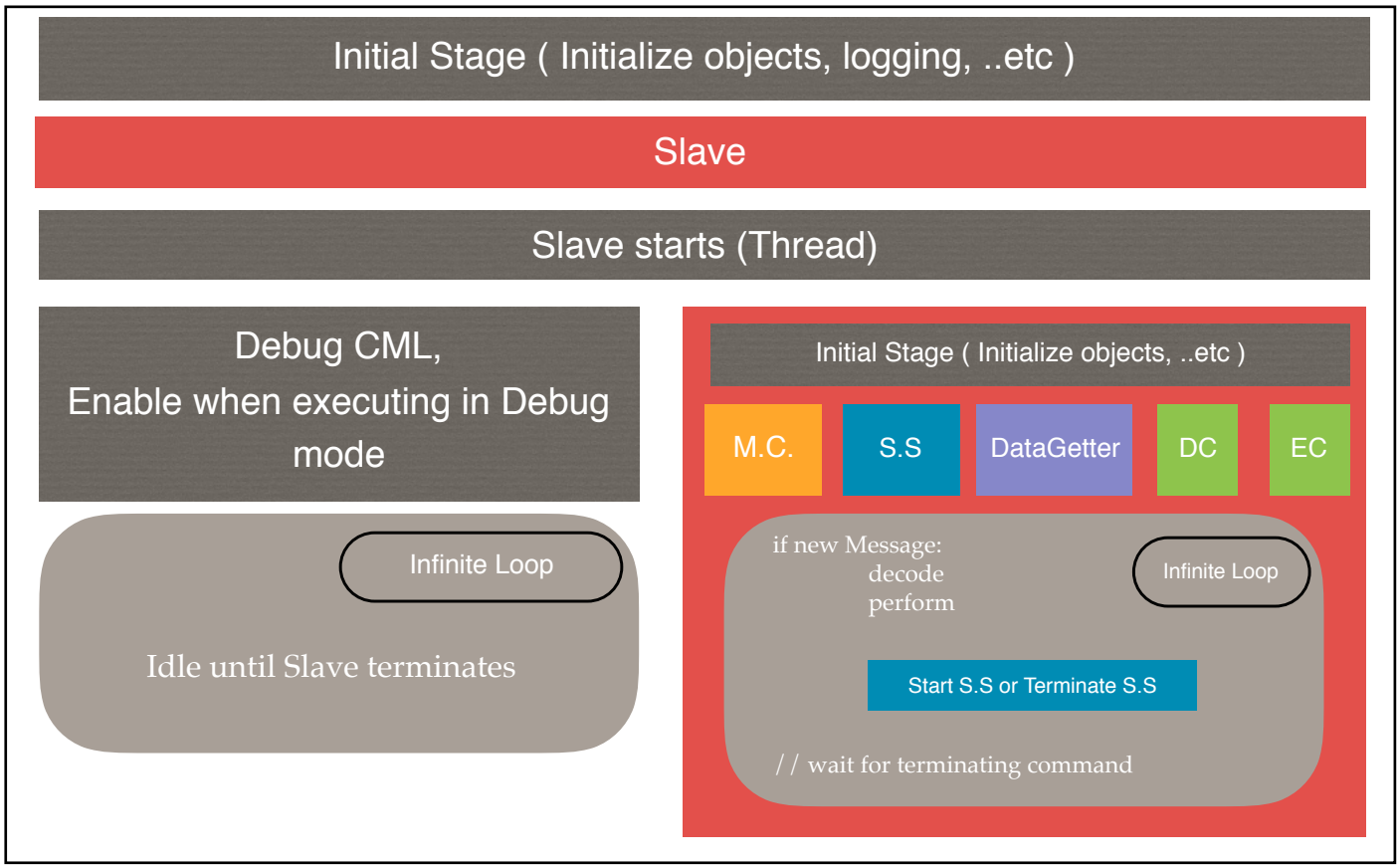
1. We limit the connection to the sqlite database. Any task that deals with the sqlite must through Database object.
2. Users and programmers should treat Database object as an interface between sqlite database and the application.
3. MessageCenter object executes as a thread, it listens the in-coming connection continuously. While it accepts the connection, another thread is created to manage the TCP connection.

3.2.3 Slave application

On the other hand, “slave” adapts other components.

1. **DataGetter**: This object deals with the production of the information requested by “master”.
2. **Slave**: This is the entry point of “slave” application, although the explicit entry point is the script of “slave”. Due to several reasons, we design that Slave, the main application, executes as a thread.
3. **SendingStatus**: This objects is responsible for sending the right data to “master”. After “slave” receives the signal, it starts to transmit, and vice versa. It executes as a thread.

(Figure 3.2-10 shows the structure of main script of “slave”)



As the figure 3.2-10 illustrates the architecture of “slave”, there are a few crucial points listed in the following.

1. The main entry point is Slave object, which inheritances from “threading.Thread”, a python3 package supporting multi-thread execution.
2. The function of the debug mode is to control “slave” by providing a CML.
3. The S.S, Sending Status, object executes as a thread, it can be terminated or re-executed by the commands from “master”.

3.2.4 The Relationship between Encoder and MessageCenter

We can clear say that the Encoder works as a Message instance factory as we mentioned previously. With the correct arguments the Encoder generates the required Message instance. While we describes the Encoder in that way, the MessageCenter deals with everything related to the communication. This relationship is shared across the master application and the slave application. We keep this relationship immortal in this system.

The following figures(Figure 3.2-11 & Figure 3.2-12) illustrate the relationship between the Encoder and the MessageCenter.

Figure 3.2-11

1. The system provides the correct arguments to the Encoder.
2. The Encoder generate the corresponding Message instance.
3. The system passes the Message instance to the MessageCenter.
4. The MessageCenter sends the message to the destination according to the content of the Message instance.

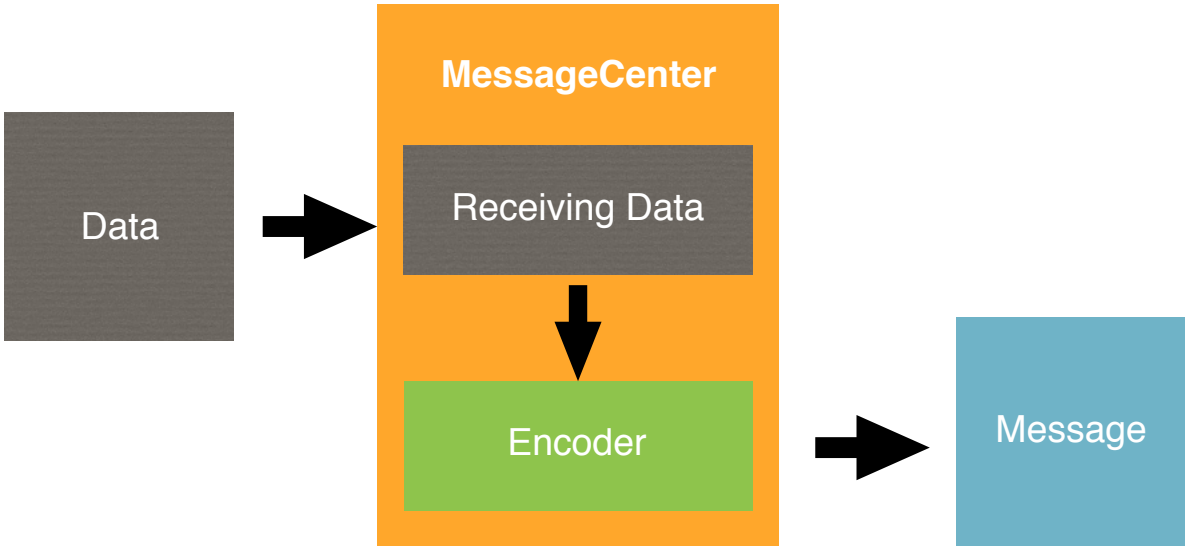


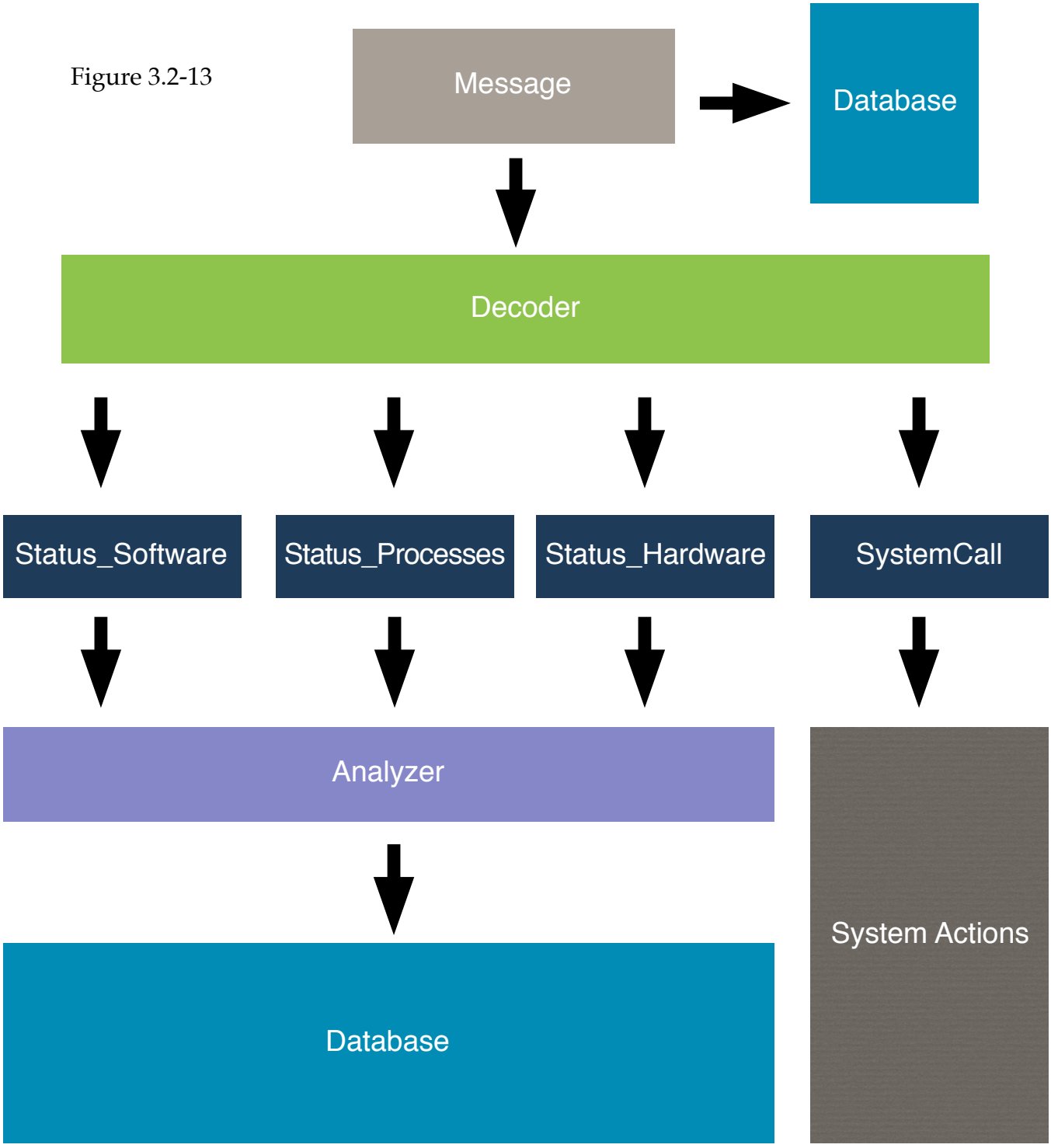
Figure 3.2-12

1. The MessageCenter of the receiver side deals with the TCP connection.
2. Collect the data from TCP connection.
3. Deliver the data to the Encoder inside the MessageCenter
4. The Encoder inside the MessageCenter generate the corresponding Message instance.
5. The system receive the Message instance from the MessageCenter.

3.2.5 The Relationship between Decoder and receivers

The Decoder is the first component that a Message instance encounters after it leaves the MessageCenter. On the both side of the applications, the Decoder decodes the Message instance to several types of instances, Status_Hardware, Status_Software, Status_Processes, and SystemCall. After the application receives those types of instances, it starts to perform a wide variety of actions. **The following figure(Figure 3.2-13) illustrates the relationship in the master application.**

Figure 3.2-13



3.2.6 Analyzer

Analyzer plays a critical role in the master application. Every monitoring information has to go through the Analyzer. The Analyzer can be easily extended in the future with the additional functions that are implemented in the system. The most important thing about the Analyzer is it makes the function of Alarming become possible. The mechanism of the function of Alarming is elaborated in the following text.

First of all, we design a data structure to store the required information of performing alarming. (Figure 3.2-14)

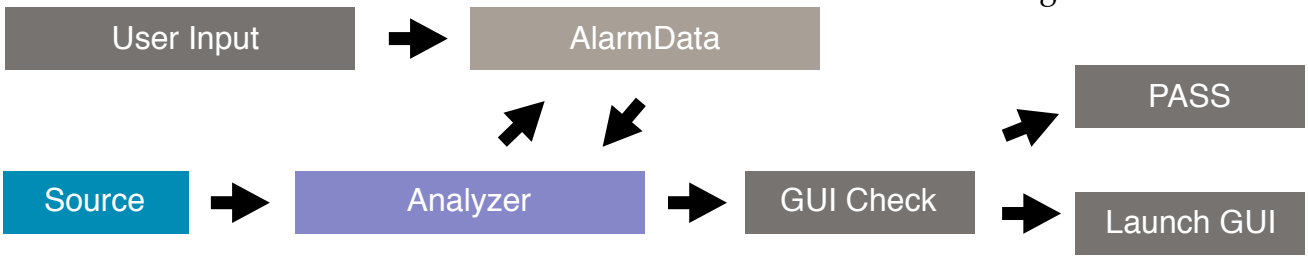
(Figure 3.2-14)
Class AlarmData
name:Srting
cpu_limit:float
mem_limit:float
cpu_count_limit:int
mem_count_limit:int
cpu_count:int
mem_count:int

a. “name” indicates the name of monitored machine.
b. “cpu_limit” indicates the percentage limitation of the cpu usage.
c. “mem_limit” indicates the percentage limitation of the Memory usage.
d. “cpu_count_limit” indicates the limitation of the counter of the cpu usage, which is introduced later.
e. “mem_count_limit” indicates the limitation of the counter of the Memory usage, which is also introduced later.
g. “cpu_count” indicates the current number of the cpu counter.
h. “mem_count” indicates the current number of the Memory counter.

Secondly, the mechanism is divided into a few steps. (Figure 3.2-15)

1. The user inputs the limitations.
2. The Analyzer analyzes the source to determine whether the current usage exceeds the limitation.
3. If so, the Analyzer marks on the AlarmData instance; thus, the counter pluses one.
4. After dealing with the data, the Analyzer checks the number of the counter to insure the number does not exceed or equal to the limitation.
5. If the number does exceed the limitation, the Analyzer actives the alarm GUI.
6. The analyzer makes sure that the alarm GUI does not show repeatedly. Thus, not until the user dismisses the alarm, the function of alarming is not available for the same machine.

Figure 3.2-15



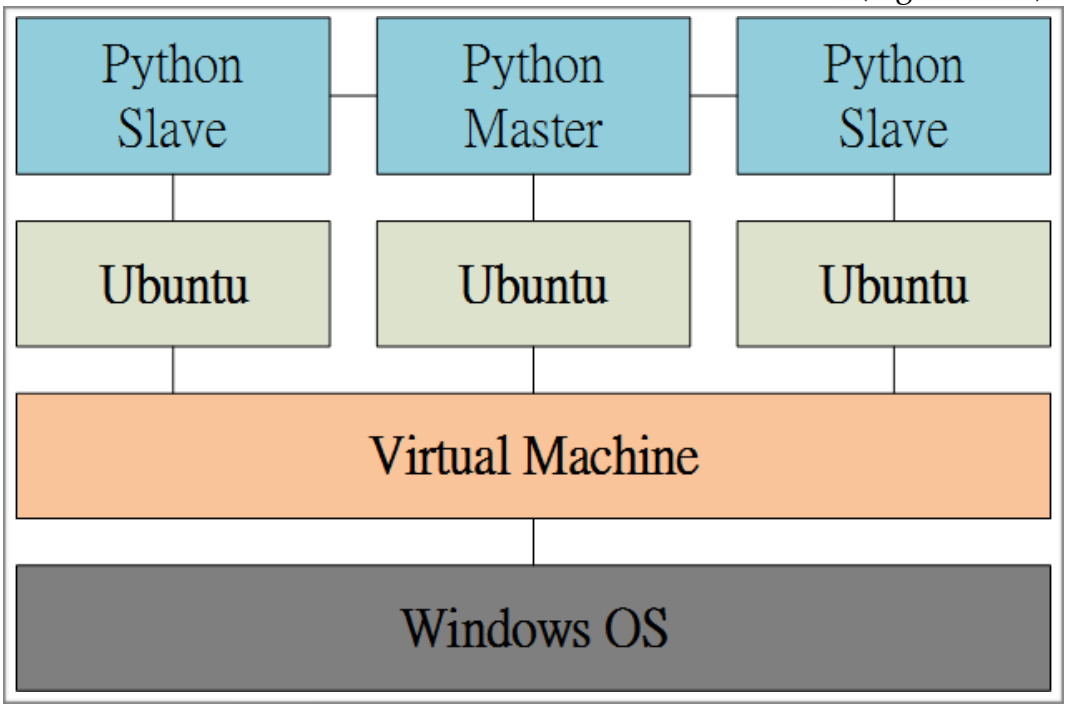
3.3 Development Environment

To implement, we use Python3 for basic system design. Also, we use a lot of modules provided by Python3 such as socket, threading, traceback, queue, pickle, logging, datetime, sqlite3, time, re and sys. For process handling and terminal status inspecting, we installed a package, psutil, to do all the jobs. We also installed PyQt5 for GUI development.

It is our first time to construct a project with this scale by using Python. We do trouble with some syntax problems. Due to the dynamic binding, a characteristic of Python, we have to manage our variables with care. Also, given that Python is a scripting programming language, the syntax itself is not as rigorous as Java. We spend some time on performing the quality assurance to avoid critical error as well as on making our coding standards.

3.4 Testing Environment

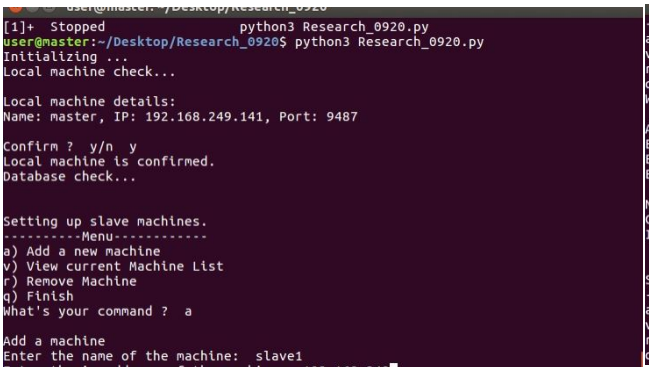
(Figure 3.4-1)



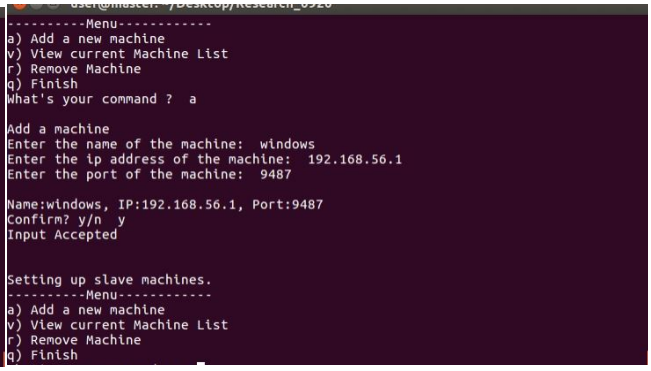
We test our program on multiple virtual machines using Ubuntu operating systems which base on Linux operating system. We also tend to make this system more compatible to different operating systems like Windows, since we design this system in Python3 so we think that it should be able to run on different operating system without adjusting the code, despite that, we test our program mainly on Linux. (Figure 3.4-1)

3.5 Demonstration in Debug-mode

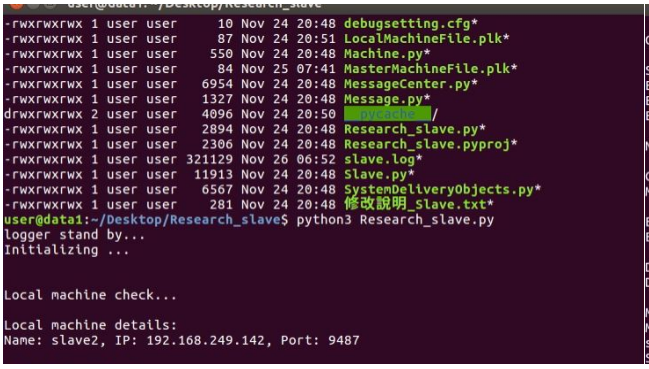
3.5.1 Installation



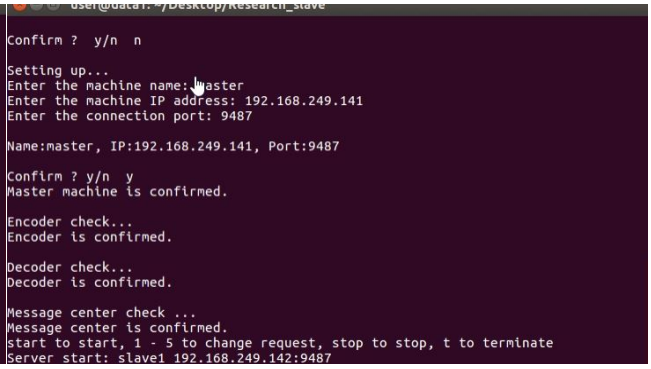
Configuring the master application,
1. Confirm the local machine



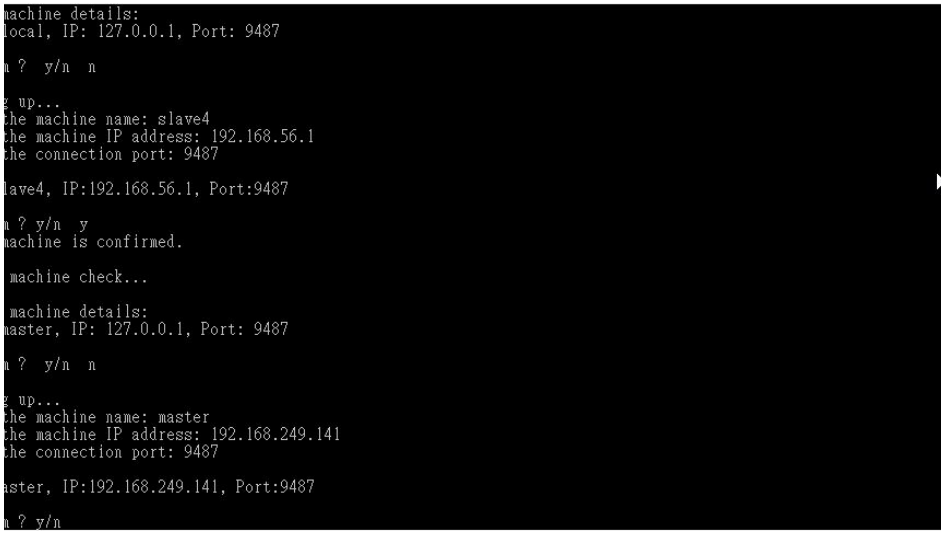
Configuring the master application,
2. Edit the slave machines list



Configuring the slave application,
1. Confirm the local machine

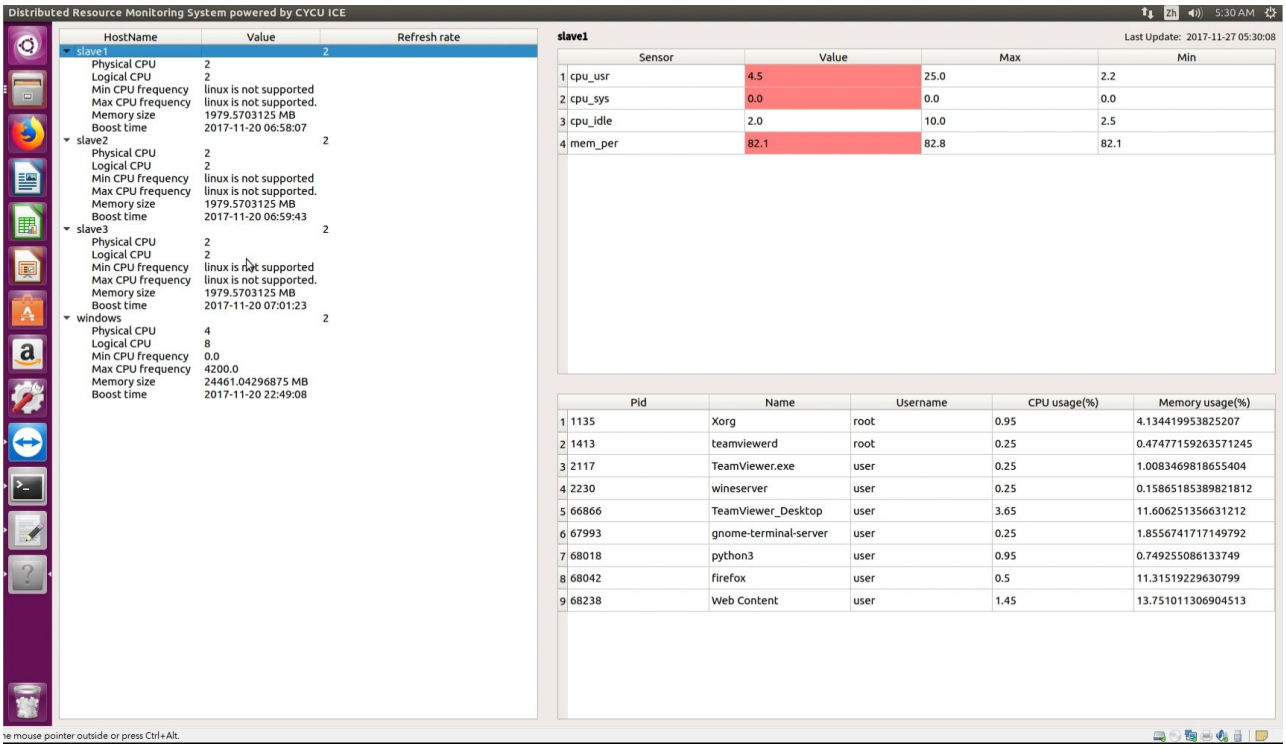


Configuring the slave application,
2. Confirm the master machine

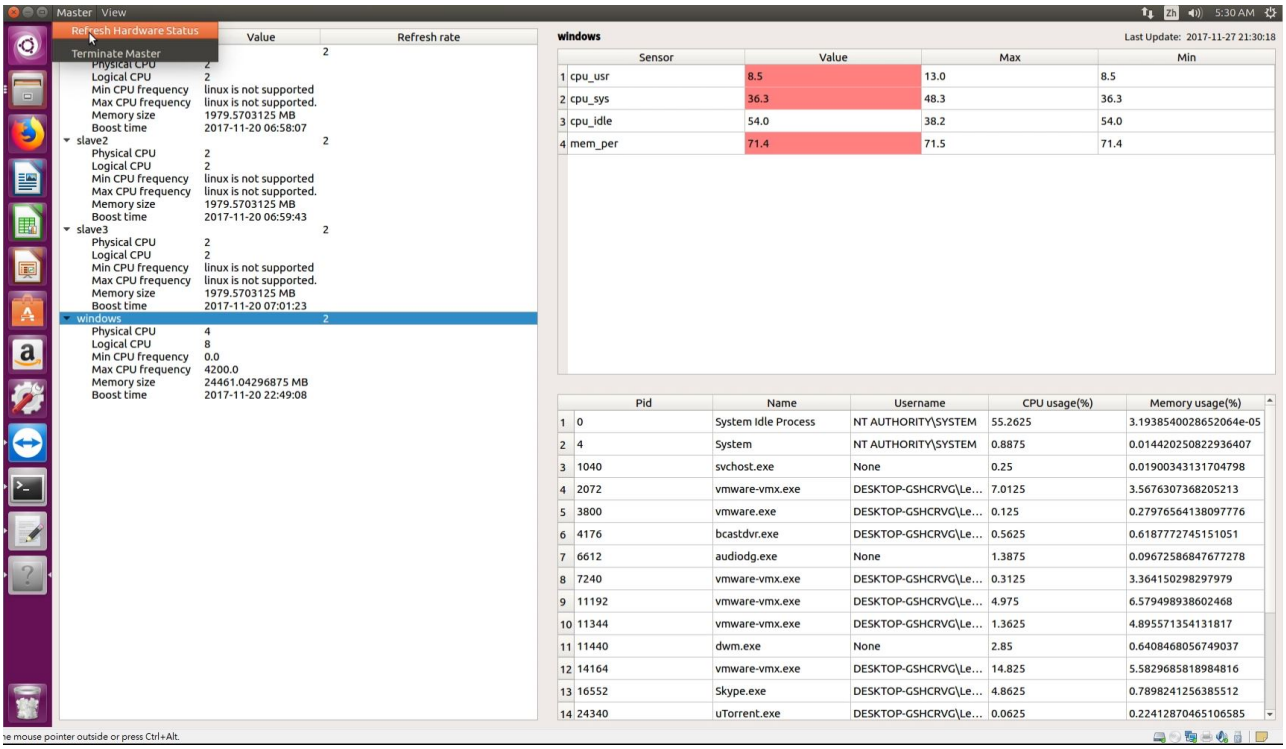


Configuring the slave application on Windows

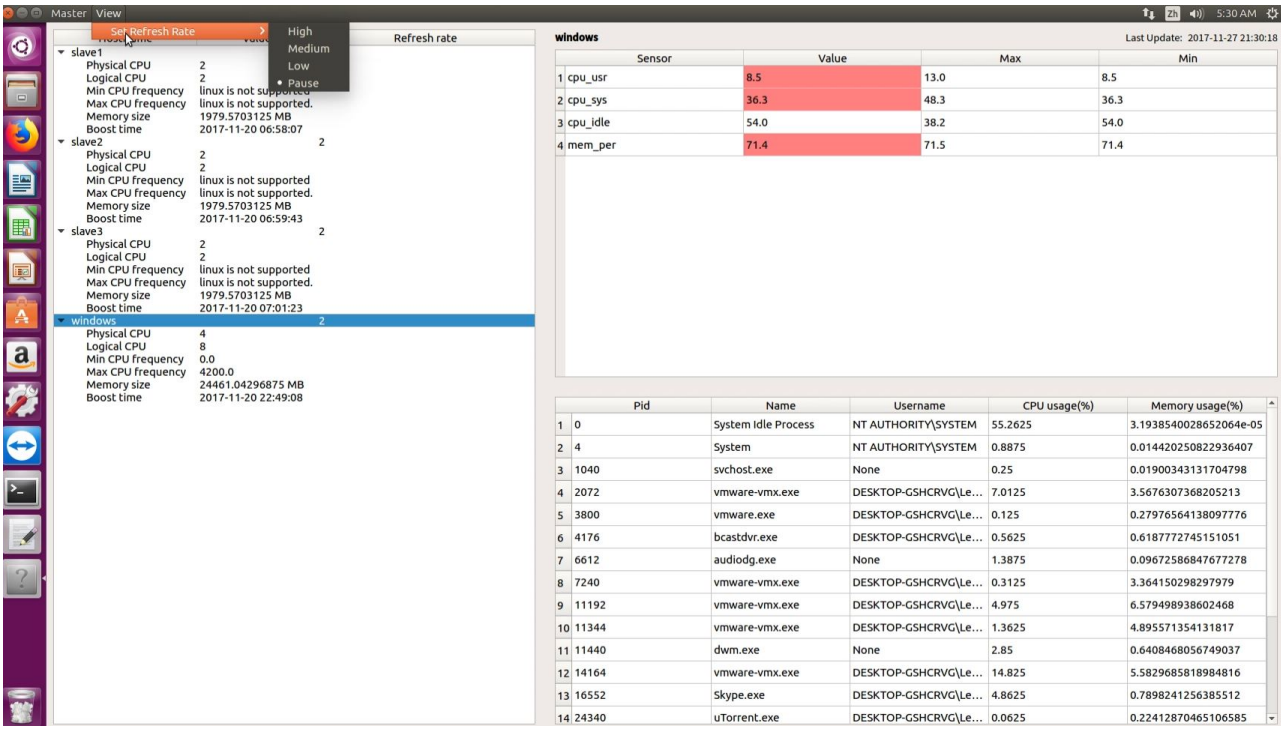
3.5.2 Main GUI



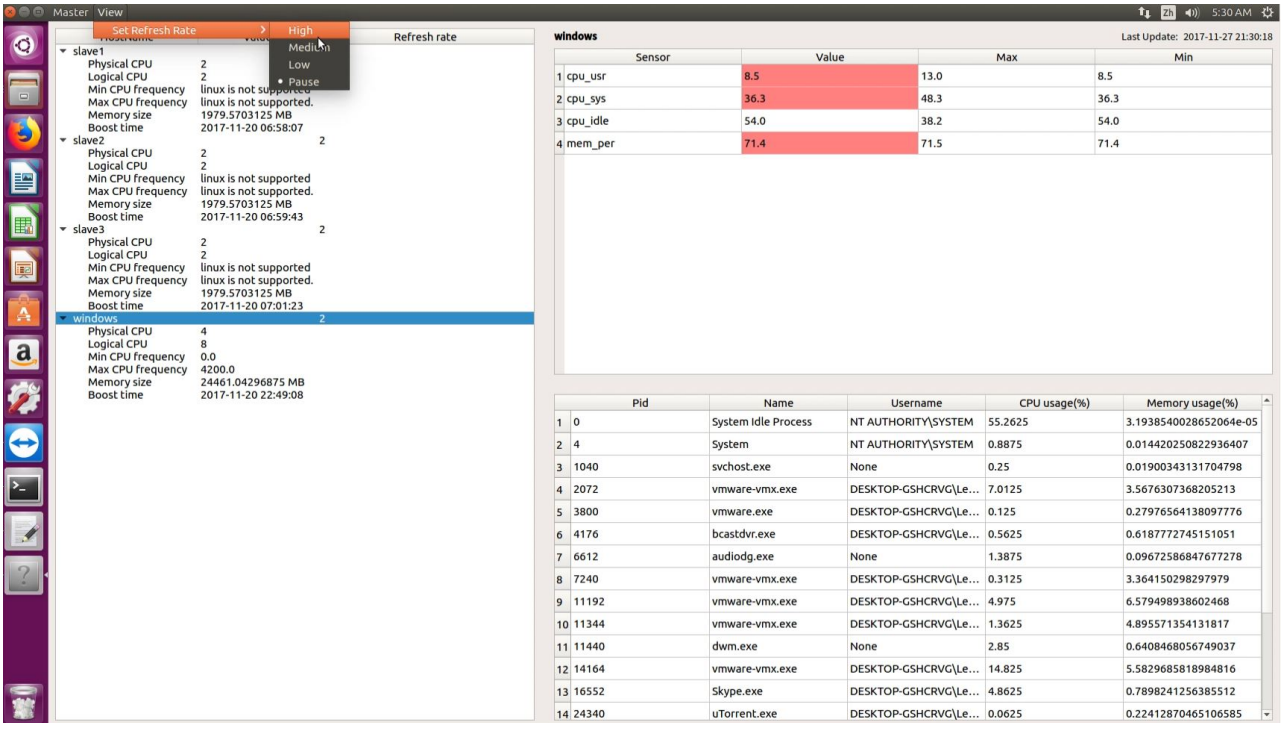
3.5.3 Master Tool Bar



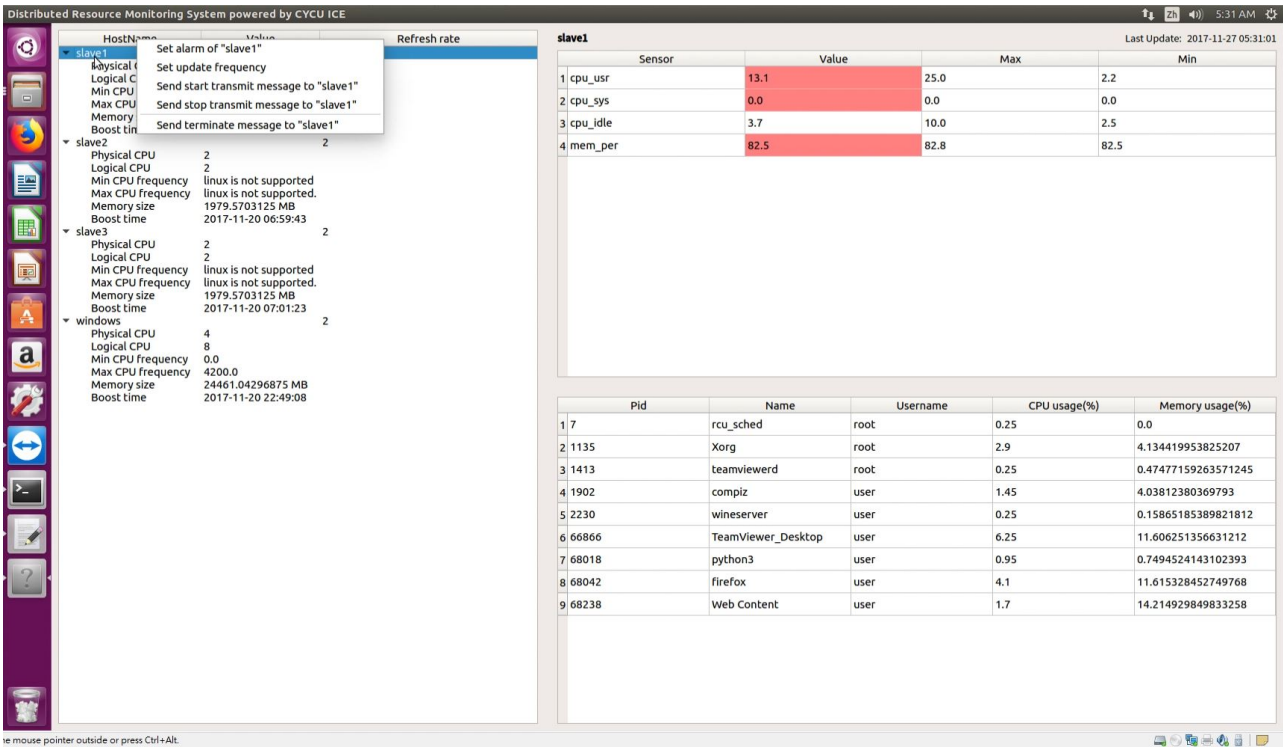
3.5.4 View Tool Bar



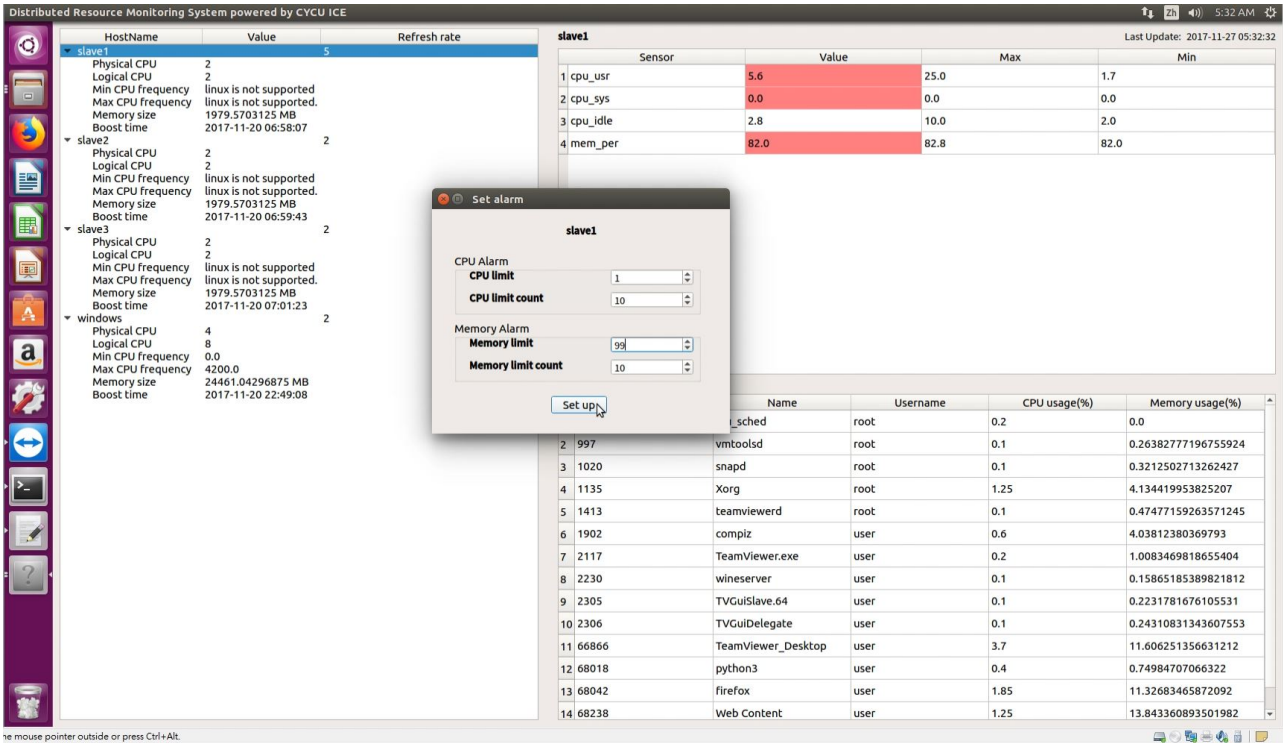
3.5.5 Change Refresh Rate



3.5.6 Slave Right Click



3.5.7 Set Alarm threshold

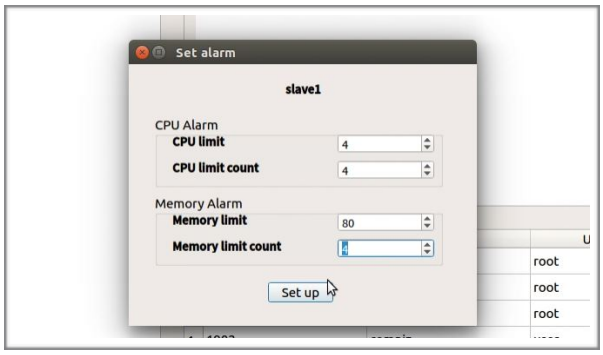


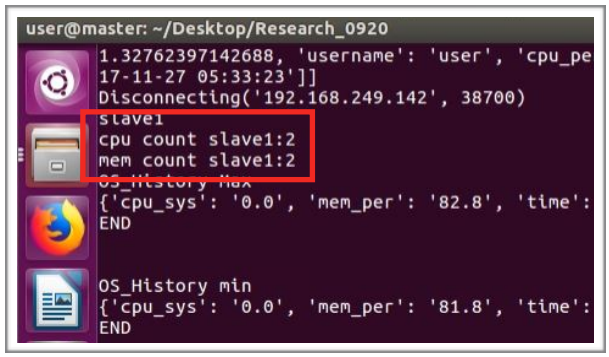
3.5.8 Highlight Table

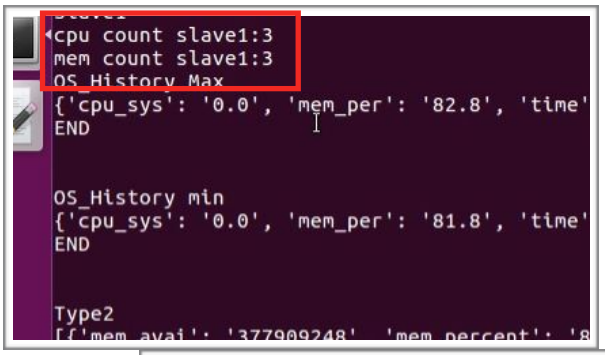
Sensor	Value	Max	Min
	5.8	25.0	1.7
	0.0	0.0	0.0
	3.0	10.0	2.0
	82.0	82.8	82.0

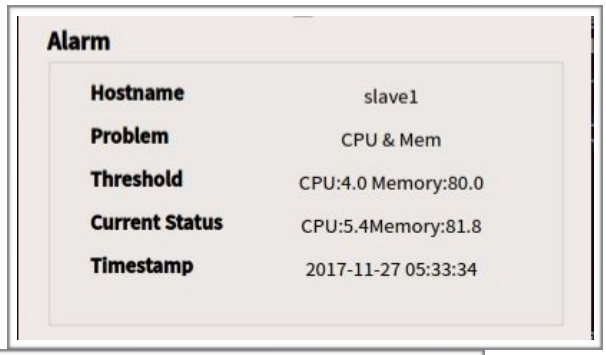
Sensor	Value	Max	Min
	6.2	25.0	1.7
	0.0	0.0	0.0
	2.3	10.0	2.0
	81.8	82.8	81.8

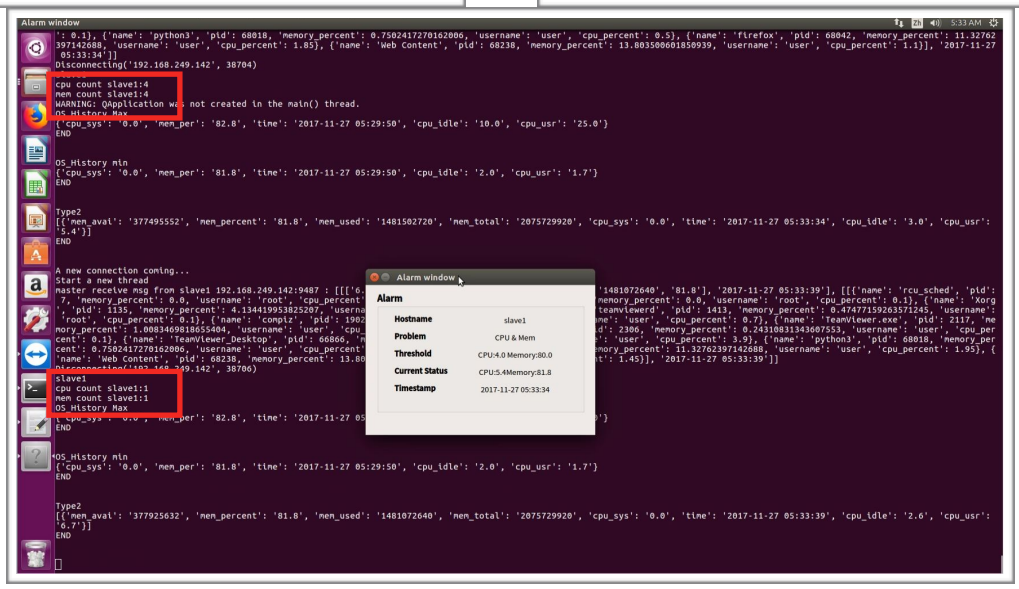
3.5.9 Alarm Function



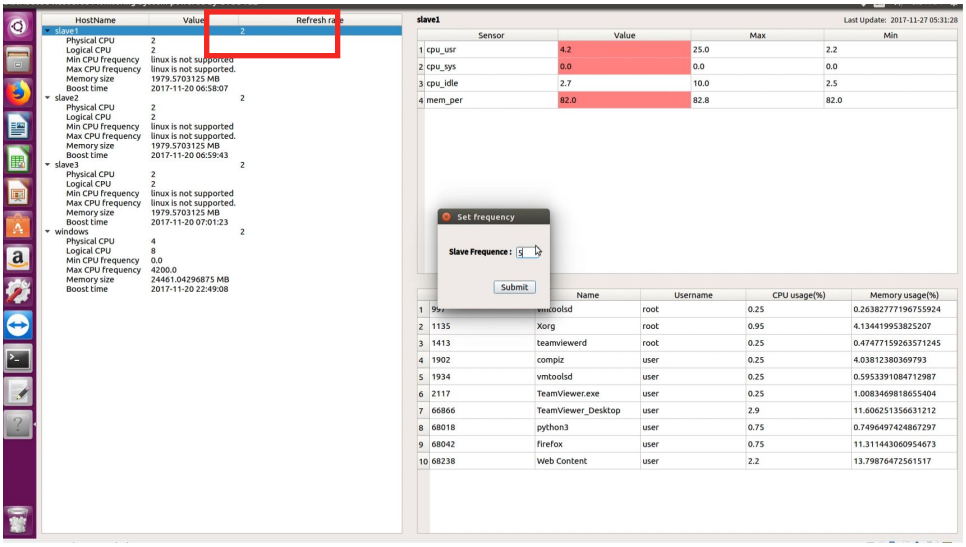


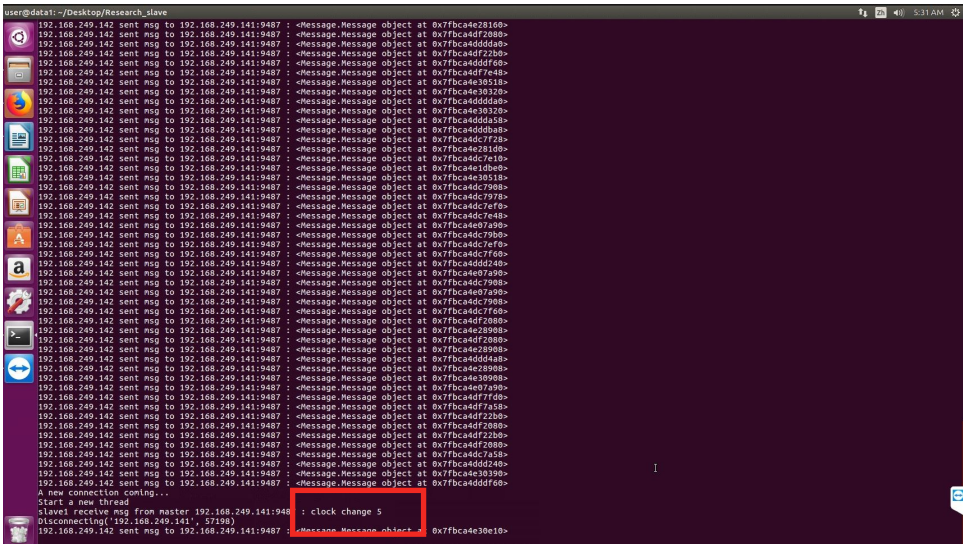


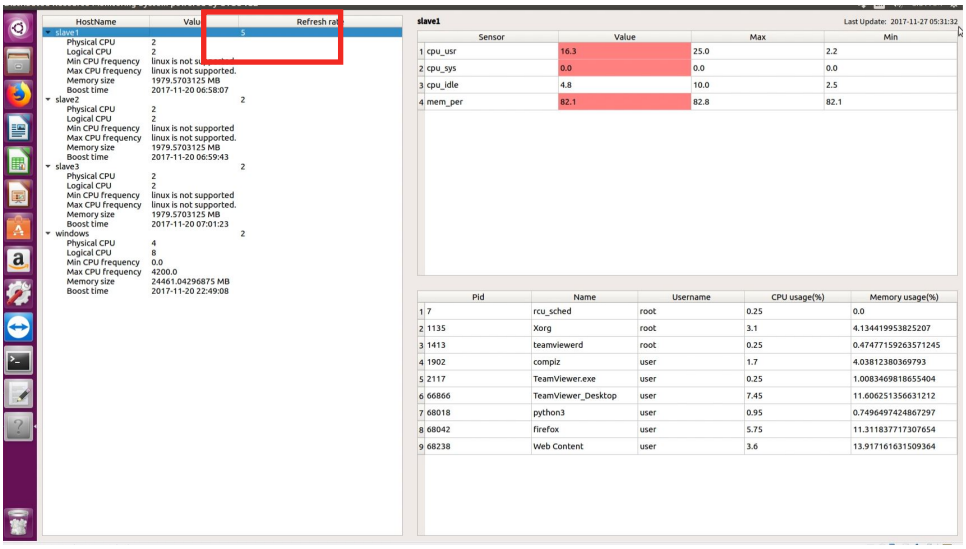




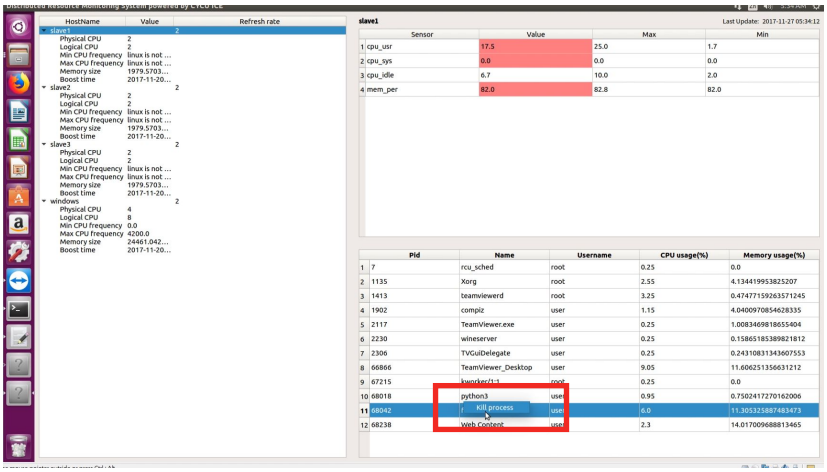
3.5.10 Set Update Frequency

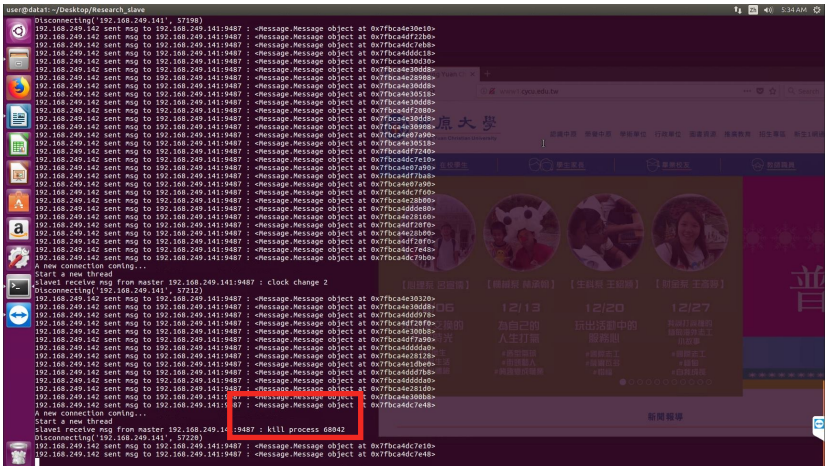


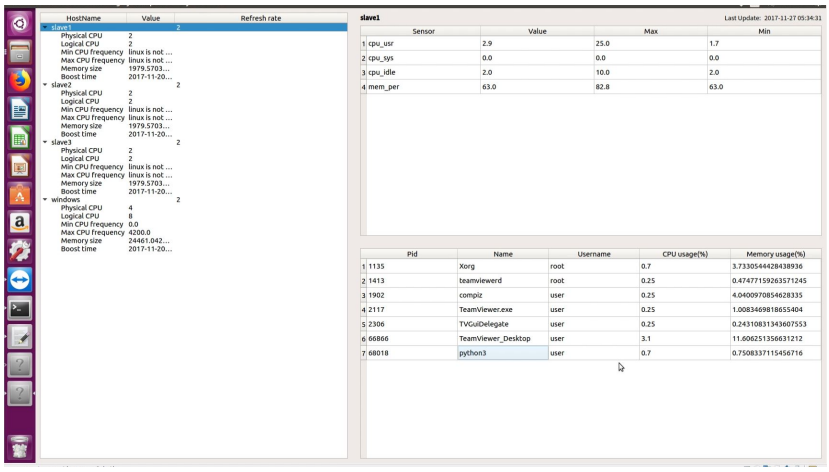




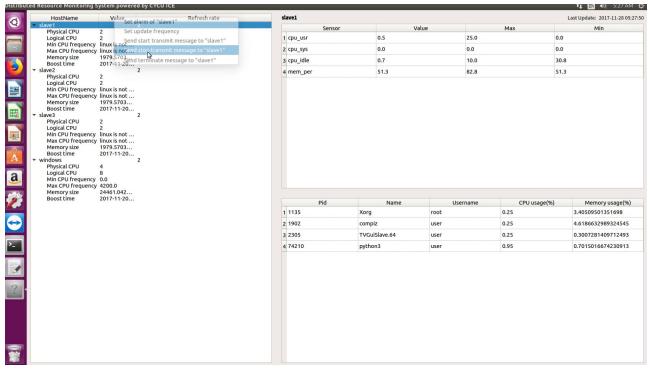
3.5.11 Kill Processes

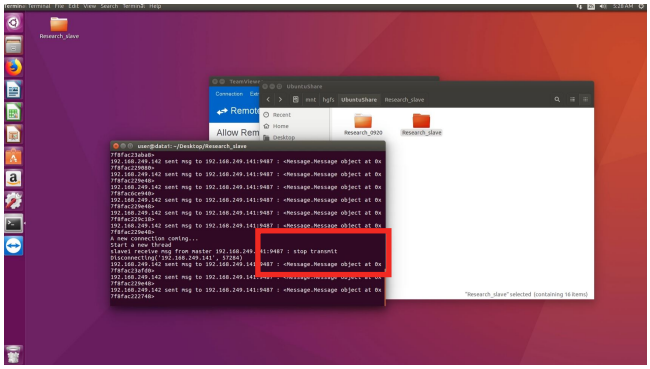




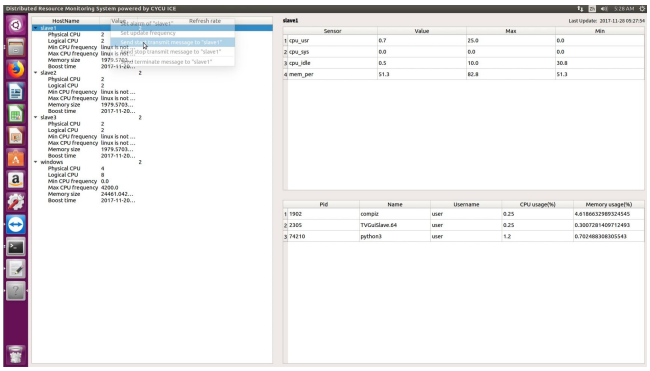


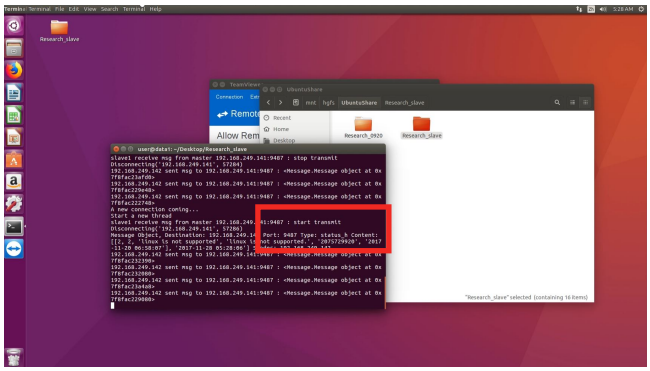
3.5.12 Stop Slave(2 pic, seventh page)



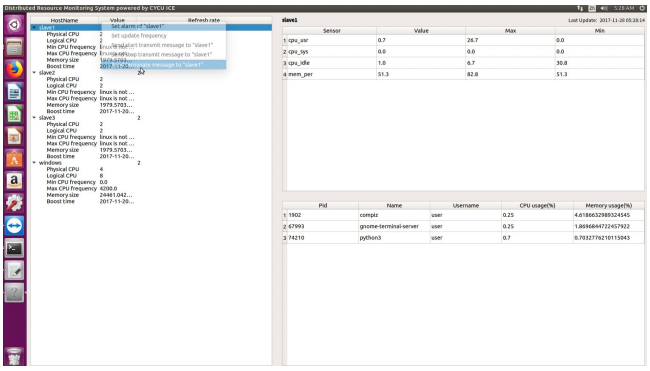


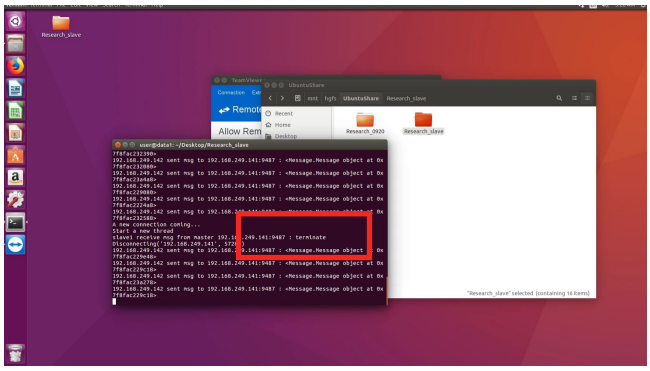
3.5.13 Start Slave





3.5.14 Terminate Slave(2 pic, seventh page)





3.6 Manual

3.6.1 User Configuration

Users shall configure the network setting in the debug mode for the first time of using the system. Users can modify the configuring file by reading it as a text file. (Figure 3.6-1) This file shall be placed in the same directory with the application.

Figure 3.6-1 shows the content of the configuring file.

In debug mode, the console(,or shell) shows the CML to configure the required setting. Users have the responsibility to make sure the information inputed into the system is correct. The following shows the examples of both applications, “master” and “salve”, respectively.

For “slave”:

1. First thing to do is to configure the local machine setting.(Figure 3.6-2)
2. After the local machine setting, configure “master” by input the information of “master”.(Figure 3.6-3)
3. It is normal that the console(,or shell) shows initializing text as well as the CML. Once CML shows, the application is ready to be part of the system. Users can re-executed it in normal mode instead of debug mode.(Figure 3.6-4)

For “master”:

1. First thing to do is also to configure the local machine setting.(Figure 3.6-5)
2. Next, a simple interface shows up, which is for managing the slave machines. There are a few functions provided by the CML. Users can select the function to manage. (Figure 3.6-6 - Figure 3.6-8)

Figure 3.6-2

Figure 3.6-3

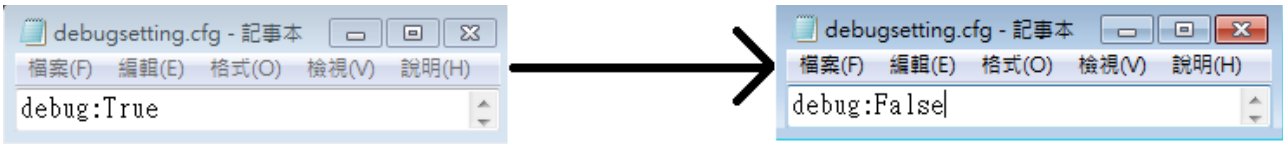


Figure 3.6-4

```
Setting up...
Enter the machine name: master
Enter the machine IP address: 192.168.56.1
Enter the connection port: 9487

Name:master, IP:192.168.56.1, Port:9487

Confirm ? y/n  y
```

Figure 3.6-5

```
Setting up slave machines.
----------Menu------------
a) Add a new machine
v) View current Machine List
r) Remove Machine
q) Finish
What's your command ?  v
('slave1', '192.168.249.141')
('slave2', '192.168.249.142')
('slave3', '192.168.249.143')
```

Figure 3.6-6

```
Setting up slave machines.
----------Menu------------
a) Add a new machine
v) View current Machine List
r) Remove Machine
q) Finish
What's your command ?  r
('slave1', '192.168.249.141')
('slave2', '192.168.249.142')
('slave3', '192.168.249.143')
Enter the machine name to remove it or input q to quit  slave3
('slave1', '192.168.249.141')
('slave2', '192.168.249.142')
Enter the machine name to remove it or input q to quit
```

Figure 3.6-7

```
Setting up slave machines.
----------Menu------------
a) Add a new machine
v) View current Machine List
r) Remove Machine
q) Finish
What's your command ?  a

Add a machine
Enter the name of the machine:  slave3
Enter the ip address of the machine:  192.168.249.143
Enter the port of the machine:  9487

Name:slave3, IP:192.168.249.143, Port:9487
Confirm? y/n  y
```

Figure 3.6-8

3.6.2 Sqlite Setting

As we mentioned above, this system adapts sqlite database to store the data. The ".db" file, generated by "master" itself should be place at the same directory. While "master" cannot connect to ".db" file, it creates a new one.

There are five tables to store the data.

1. Table "machine_list" stores the information of slave machines, its name, its ip address, its connection port, and its object in binary formation. (Figure 3.6-9)

   The SQL command is "'CREATE TABLE machine_list (name TEXT PRIMARY KEY NOT NULL, ip TEXT NOT NULL, content BLOB NOT NULL)'"

Table machine_list (Figure 3.6-9)			
name TEXT	ip TEXT	port TEXT	content BLOB

2. Table "message_record"stores the records of send/received message, the time, the sender, the receiver, and the content.(Figure 3.6-10)

   The SQL command is "'CREATE TABLE message_record (time TEXT PRIMARY KEY NOT NULL, sender TEXT NOT NULL, receiver TEXT NOT NULL, content TEXT NOT NULL)'".

Table message_record (Figure 3.6-10)			
time TEXT	sender TEXT	receiver TEXT	content TEXT

3. Table "status_record" stores the status of slave machines, including the deleted ones. (Figure 3.6-11)

   The SQL command is "'CREATE TABLE status_record (time TEXT PRIMARY KEY NOT NULL, owner text NOT NULL, type TEXT NOT NULL, content BLOB NOT NULL)'"

Table status_record (Figure 3.6-11)			
name TEXT	owner TEXT	type TEXT	content BLOB

4. Table “os_history” stores the OS status which is the most significant status, such as the highest cpu usage. (Figure 3.6-12)

   The SQL command is “'CREATE TABLE os_history (time TEXT PRIMARY KEY NOT NULL, owner text NOT NULL, cpu_usr TEXT NOT NULL, cpu_sys TEXT NOT NULL, cpu_idle TEXT NOT NULL, mem_per TEXT NOT NULL)’”

Table os_history (Figure 3.6-12)					
time TEXT	owner TEXT	cpu_usr TEXT	cpu_sys BLOB	cpu_idle TEXT	mem_per TEXT

5. Table "alarm_data" stores the alarming data of each monitored machine. (Figure 3.6-13)

   The SQL command is "'CREATE TABLE alarm_data (name TEXT PRIMARY KEY NOT NULL, cpu_limit TEXT NOT NULL, mem_limit text NOT NULL, cpu_count_limit TEXT NOT NULL,mem_count_limit TEXT NOT NULL)'"

Table alarm_data (Figure 3.6-13)				
name TEXT	cpu_limit TEXT	mem_limit TEXT	cpu_count_limit TEXT	mem_count_limit TEXT

The binary data, formatted in BLOB, can be loaded by using “pickle", one of the python3 package.

Part 4 Future work

We are more than convinced that the potentiality of this system is tremendous. Not only complementing this system is a possible task but also to expense this system into a project with a large scale, such as developing a Distributed-Computing platform, which is what we plan to do at first, is likely. We have considered a number of possibilities as well as some considerations.

4.1 Short-Term Works

We believe that there exist a few possible short-term plans to improve this project.

HCI, Human-Computer Interface

Currently, we focus more on the machine status supervising and not much on the processes managing, we believe it is a good idea if we can concentrate on certain processes optionally. Moreover, graphical display about the resources usage was taken into consideration when we were discussing our goal.

Database and Logging Revised

As data of each slave will be saved in master's database though the data is quite small that takes only little storage, it is growing constantly while this system is working. It will, in the end, either use up the storage or take too much of it so data revising is definitely a must-do thing if we want to run the system for a long time.

Daemon

Making it as a service is a practical idea, which means running the daemon in the background on Linux-Based OS. Currently, we still need to launch this system by hands to establish the connection. If we can make it into a service, then it will be much more friendly to new users as well as reduce the tedious routines.

Slaves monitor themselves

Make each slave monitor themselves is also a sound improvement. We may consider a scenario that operating system failed, and nothing can transmit to master. This situation is a tricky problem, how can master know that the remote slave is under OS failure since it will not send anything back to master? A solution is to create a routine table which provide the sufficient information for slaves to find another slave. For instance, an under polling slave is having an operating system failure, and the polling one will discover the failure immediately since it doesn’t ACK back. Once the polling slave detects the OS failure of another one, it can launch a trap message to the master. Therefore, the master can acknowledge the critical problems as soon as possible.

4.2 Long-Term Development

Reinforce the architecture of the system

First and foremost, we are well aware of the fact that this system is not perfect. Besides the system evolution mentioned in the previous section, the improvement of the human-computer interface, known as HCI, there exist some defects of the system. For example, the insurance of data deserves our attention. We may either make the data store with a distributed way or make a backup of the database in somewhere. Is it possible to apply HDFS in our system? If so, how can we combine this technology properly and efficiently? These problems are worthy to dissolve.

Connect to the area of IoT

Furthermore, this system not only can be applied on the normal computer system but the embedded system as well. In the area of IoT, Internet of Things, the importance of the embedded system cannot be overemphasized. The warehouse system of Amazon Inc. is one of the most successful examples. A useful monitoring system is the most critical part. We can state confidently that our project is likely to be developed into this scale by improving the HCI as well as the functions of "slave". In addition to a characteristic of Python3, Python3 is a cross-platform programing language; therefore, the compatibility should not be a problem.

Become to a Cluster Management Tool

The last but not the least, on the other hand, we may expend the function and the responsibility of “master”. Anaconda is one of the most powerful cluster management platform. This system can be our goal. After experience the functions of Anaconda, we realize that Distributed Resource Monitoring System still needs lots of effort to make it become a practical utility.

Make a Distributed-Computing Platform

Lastly, we believe that we finish the fundamental layer of Distributed-Computing platform, the layer of the communication. An idiosyncrasy of Distributed-Computing is each slave may interact with not only master machine but also slave machine. We believe we can have the system acquire this function in a flash. However, a cluster management is required to operate a Distributed-Computing platform. The algorithm of Map-Reduce, an algorithm introduced by Apache Hadoop, as well as the Apache Spark Distributed-Computing platform are required a task management. We assert that the efficiency of a system corresponds to the ability of automatically managing tasks in a distributed computing environment. At this moment, “master” has the ability to monitor the status of its slaves. In other words, we can enhance the ability of analyzing data to improve its productivity. This is a huge bottleneck for us since we notice that the knowledge that we have now is not sufficient. A task management is the next thing to do on the way of developing a Distributed-Computing platform.

Part 5 Summary

Distributed Resource Monitoring System

We, subject to the limited time, have been changing our targets on our way working on the projects, despite that, we still stick to the basis of distributed system architecture. We have studied lots of existing tools and applications such as Anaconda, Hadoop, and Spark, although we do not make use of all of them but we certainly learn something. After all that, we finally decide to set our goal, Distributed Resource Monitoring System. We have been changing the tools we use to accomplish what we want or to get closer to our goal since there are various ways to approach; we always try to pick the best ones for us to utilize.

Although we have made something that seems doing the jobs we offer but actually we have much more ideas that we want to feature in this project, the ideas require more intricate settings and will take undoubtedly much more time to implement without a doubt.

We believe this project has huge potential for further development, since the trend indicates that things like IoT, Internet of Things, and IoV, Internet of Vehicles, are getting more important, this project can definitely be an aid for us to comprehend what the people in the globe are doing in this realm and we know there are much more than that.

Software Engineering

Software Engineering is also a crucial concern, even though this subject does not obtain much attention during the period of undergraduate studies. In the development of Distributed Resource Monitoring System, we are attempting to develop this system with a high maintainability as well as make this project follow the principle of Software Engineering. We adapt Object-Oriented Programing for the sake of teamwork and maintainability. We adapt the incremental development model mentioned in the reference book for the sake of the fluency of the project development. We manage the schedule of the project with the Gantt Chart. We make a brief document to describe the system. We can confidently state that the practices of Software Engineering are not only precious for present but also useful in the future.

Speaking of the teamwork, the heterogeneity of the ability of the students has a great influence on the productivity of the team. In other words, the fact that assigning a suitable work for each individual matters a lot, when it comes to the efficiency. In the past ten months, we have encountered a few conflicts; after each conflict, we do learn something from it and improve in the next time. Although every team member is responsible for distinct tasks, we do acknowledge that to achieve the goal requires everyone to finish his/her task on time, even with tiny mistakes.

Special Thanks

It is our pleasure to have Mr. Ke be our project advisor. Mr. Ke provides lots of advice with his professional aspects while we encounter bottlenecks not only during the period of surveying but also in the process of the implementation. Doubtlessly, Mr. Ke is the one who play an important role in this project.

Part 6 Reference

Links

https://stackoverflow.com/questions/41026842/python-pass-more-than-two-arguments-in-signal
https://stackoverflow.com/questions/70528/why-are-pythons-private-methods-not-actually-private
https://stackoverflow.com/questions/2728999/how-to-get-top-5-records-in-sqlite
https://stackoverflow.com/questions/482410/how-do-i-convert-a-string-to-a-double-in-python
https://stackoverflow.com/questions/3845362/python-how-can-i-check-if-the-key-of-an-dictionary-exists
https://stackoverflow.com/questions/1602934/check-if-a-given-key-already-exists-in-a-dictionary
https://stackoverflow.com/questions/13861594/valueerror-invalid-literal-for-int-with-base-10
https://stackoverflow.com/questions/32659993/what-is-the-difference-between-incremental-model-and-prototyping-model
https://stackoverflow.com/questions/18614665/run-python-in-terminal-and-dont-terminate-it-when-terminal-is-closed
http://interactivepython.org/runestone/static/thinkcspy/SimplePythonData/StatementsandExpressions.html
https://docs.python.org/3/library/sqlite3.html
https://docs.python.org/3/library/pickle.html
https://docs.python.org/2/library/signal.html#module-signal
http://www.sqlitetutorial.net/sqlite-python/update/
http://psutil.readthedocs.io/en/latest/
https://en.wikipedia.org/wiki/RSA_(cryptosystem)
https://www.datacamp.com/community/tutorials/apache-spark-python#gs.G6M1h70
https://www.datacamp.com/community/tutorials/apache-spark-python
https://spark.apache.org/docs/0.9.1/python-programming-guide.html
http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_setup/py_intro/py_intro.html
https://pythonprogramming.net/loading-video-python-opencv-tutorial/
https://stackoverflow.com/questions/41441150/how-to-read-video-files-using-python-opencv

Books

Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing by Frank Buschmann, Kevin Henney, Douglas C. Schmidt
ISBN-13: 978-0470059029

Python+Spark 2.0+Hadoop機器學習與大數據分析實戰 by 林大貴
ISBN-13:978-986-434-153-5

Software Engineering (9th Edition) by Ian Sommerville
ISBN-13: 978-0137035151