



Universidad Nacional Experimental de Guayana

Área de Informática

Buscador de Archivos (File Finder)

Etcheverry, R. y García, I.

15/06/2016

Sistemas Distribuidos. 2016-I

TABLA DE CONTENIDO

RESUMEN	5
INTRODUCCIÓN	6
REVISIÓN TEÓRICA	8
Mensaje	8
Cola de mensajes.....	8
SnakeMQ.....	8
Buscador de archivos	8
OBJETIVOS DE LA INVESTIGACIÓN	9
Objetivo General	9
Objetivos Específicos.....	9
METODOLOGÍA DE DESARROLLO	10
Etapas del modelo utilizado	11
Análisis de requerimientos.....	11
Diseño.....	11
Codificación	12
Pruebas.....	17
Implementación	17
HARDWARE UTILIZADO	18
Arquitectura del sistema	19
Estructura de las colas de mensajes.....	20
Modelo de interacción	20
Modelo de fallas.....	23
Error de IP.....	23
Eliminación de nodos	23
Detección de nodos: Proceso de incorporación y desincorporación de nodos	24
Instalación	24
Pasos para configurar el Buscador de archivos:.....	26
Características comunes para servidor_1 y servidor_2:	28
Búsqueda de archivos desde el cliente (master).....	28
PRUEBAS Y RESULTADOS.....	30

Búsqueda de un archivo basado en su extensión	30
Búsqueda de un archivo basado en su nombre y extensión.....	30
Búsqueda de dos archivos simultáneamente (conurrencia) desde un mismo cliente.....	31
Visualización de un archivo a través del navegador	33
CONCLUSIÓN	35
BIBLIOGRAFÍA	36

LISTADO DE ILUSTRACIONES

Ilustración 1 Metodología en Cascada	10
Ilustración 2 Etapa Codificación, Función obtener IP equipos (servidor).	12
Ilustración 3 Etapa Codificación, Función para iniciar Servicio Web y función para servir archivo solicitado (servidor/slave).....	12
Ilustración 4 Etapa Codificación, Función llamar al navegador con el archivo solicitado (cliente/master)	13
Ilustración 5 Etapa Codificación, Función permisología configuración de carpetas (servidor)	13
Ilustración 6 Etapa Codificación, Función Buscar Archivos (servidor)	14
Ilustración 7 Etapa Codificación, Función Ventana Principal Cliente.....	15
Ilustración 8 Etapa Codificación, Función Ventana Configuración Cliente	16
Ilustración 9 Etapa Codificación, Función guardar host configurados (Cliente)	16
Ilustración 10 Etapa Codificación, Función archivo encontrado (Cliente)	16
Ilustración 11 Diagrama Arquitectura del Sistema	19
Ilustración 12 Estructura cola de mensaje	20
Ilustración 13 Diagrama Caso de Uso.....	20
Ilustración 14 Diagrama de secuencia para configuración inicial	21
Ilustración 15 Diagrama de secuencia para la búsqueda de archivos	22
Ilustración 16 Modelo de fallas: Error de IP	23
Ilustración 17 Modelo de fallas: Eliminación de nodos	23
Ilustración 18 Modelo de fallas: Proceso de incorporación y desincorporación de nodos	24
Ilustración 19 Instalación, Configuración de permisología	26
Ilustración 20 Instalación, Configuración Host desde el Cliente	26
Ilustración 21 Instalación, Configuración Servidor_2 (Slave).....	27

Ilustración 22 Instalació, Servidor_1 (Slave)	27
Ilustración 23 Instalación, Comando para ejecución de servidores y cliente	28
Ilustración 24 Instalación, Ventana Cliente (Master).....	29
Ilustración 25 Prueba Servidor_2, Búsqueda archivo *.c.....	30
Ilustración 26 Prueba, ubicación archivo a buscar documentoABuscar.txt.....	31
Ilustración 27 Prueba notificación de la ruta del archivo documentoABuscar.txt por parte del Servidor_2	31
Ilustración 28 Prueba de varias peticiones simultáneamente (conurrencia).....	32
Ilustración 29 Prueba, notificación de archivos en peticiones simultaneas (servidor).....	32
Ilustración 30 Prueba Ventana Cliente, resultado de peticiones simultáneamente.	33
Ilustración 31 Prueba, visualización de un archivo pdf en el navegador del cliente	34

RESUMEN

En el presente proyecto se hizo el diseño, desarrollo e implementación de un Buscador de Archivos, haciendo uso de la arquitectura de colas de mensajes, a través de la tecnología de SnakeMQ. El Buscador de Archivos al ser cliente-servidor consta de dos aplicaciones, que corren una en el cliente y otra en el servidor, la primera posee una interfaz que permite al usuario generar búsquedas, la segunda realizar la búsqueda y transmisión de peticiones dentro de los nodos servidores de archivos. Al emplear la arquitectura de cola de mensajes con SnakeMQ se pudo evidenciar y aprovechar la simplicidad que este brinda al sistema, dado que se encarga de almacenar los mensajes en colas que los nodos interpretarán con el fin de atender cada petición que se genere.

La metodología de desarrollo empleada es en cascada. Consiste en cumplir una serie de pasos secuenciales tales como: identificación de requerimientos, modelado del sistema, codificación, pruebas y finalmente la implantación, estos se expondrán en el desarrollo del informe.

En el diseño y desarrollo se evidenció el modelo de fallas, el cual hace tratamiento de los distintos casos que se pudiesen presentar tales como: errores de configuración e incorporación y desincorporación de nodos.

Algunos de los resultados más relevantes en el desarrollo de la aplicación fueron la conexión entre nodos para el envío y recepción de mensajes, respuestas asíncronas por parte de los servidores a las peticiones del cliente de los archivos solicitados y lograr que la aplicación fuera multiplataforma (Linux y Windows).

INTRODUCCIÓN

En la actualidad se almacenan un sin número de archivos en las computadoras (de forma temporal o permanente), archivos que pueden ser imágenes, videos, audios, documentos, presentaciones, hojas de cálculo, notas, entre otros, que obligan a acudir al buscador de archivos para del sistema operativo continuamente, lo cual conlleva a la pregunta ¿Se estará preparado para aprovechar al máximo todos estos archivos que se almacenan?

Se hace difícil recordar exactamente todas las rutas de todos los archivos que se almacenan, y aun si esto es posible, resulta poco eficiente. Sin embargo algunas personas mantienen una estructura ordenada de sus archivos en carpetas, de tal modo que puedan realizar la búsqueda de información con cierta facilidad. Pero de igual manera no recuerdan su ubicación, si no que utilizan su disciplina para ubicar los archivos en el momento en que estos lleguen a su computadora. Incluso de esta manera, existe la posibilidad de tener algún archivo en una ubicación errónea o que no exista, por lo cual la búsqueda será más lenta.

Estas causas conllevan al surgimiento de lo que se conoce como bibliotecas digitales, que son en principio una colección de objetos digitales más o menos organizada, que sirve a una comunidad de usuarios definida. Estas ponen a disposición de todos los usuarios los recursos que poseen.

El presente trabajo se enfoca en desarrollar una solución eficiente al proceso de buscar archivos, haciendo uso de bibliotecas digitales. Dicha solución propone un Buscador de Archivos que pueda encontrar rápidamente todo lo que esté almacenado en bibliotecas de información digital (servidores) que coincida con los datos (nombre de archivo y extensión) introducidos por el usuario final (cliente).

El Buscador de Archivos es desarrollado en software libre y puede ser corrido en plataformas de Linux y Windows, es ligero y su uso es muy intuitivo permitiendo la fácil navegación.

Está preparado para usar las convenciones de búsqueda que permiten encontrar rápidamente lo que se necesite. Como por ejemplo si se quiere encontrar un archivo de algún

programa desarrollado en java se puede colocar en la barra de búsqueda: “.java”, y así con cada extensión deseada.

REVISIÓN TEÓRICA

Mensaje

“Un mensaje (message en inglés) es una unidad de datos enviada entre dos equipos. Un mensaje puede ser muy sencillo y constar de una simple cadena de texto, o puede ser más complejo e incluir, por ejemplo, objetos incrustados” [MICROSOFT (2016)].

Cola de mensajes

Los mensajes se envían a colas. Una cola de mensajes es un contenedor que alberga mensajes mientras están en tránsito. El administrador de colas de mensajes actúa como intermediario en la transmisión de un mensaje desde su origen hasta su destino. El principal objetivo de una cola es proporcionar enrutamiento y garantizar la entrega de los mensajes; si el destinatario no está disponible en el momento de enviarse un mensaje, la cola lo guarda hasta que pueda entregarse correctamente [MICROSOFT (2016)].

SnakeMQ

De acuerdo con la página, [SNAKEMQ (2015)] “SnakeMQ es una pequeña librería de Python para la comunicación fácil y fiable entre los hosts. Sólo se tiene que enviar el mensaje y dejar que la librería se encargue de la entrega”.

Características de SnakeMQ, según la página oficial

1. Python puro, cross-platform.
2. Reconexión automática.
3. Entrega confiable
4. Asíncrono

Buscador de archivos

El Buscador de Archivos inspecciona a través de las carpetas previamente configuradas, los archivos cuyo nombre y extensión coinciden con los datos especificados al inicio.

Una vez que se haga click en el botón Buscar, y se inicie el proceso, se generará una lista exponiendo las rutas de los archivos que retornarán los servidores en el Buscador de Archivos que se registraron como coincidencia, facilitando al usuario control, acceso y reconocimiento de los mismos.

OBJETIVOS DE LA INVESTIGACIÓN

Objetivo General

Desarrollar un Buscador de Archivos haciendo uso de la arquitectura SnakeMQ, apoyándose en el lenguaje de programación Python y sus librerías de colas de mensajes.

Objetivos Específicos

1. Análisis de los requerimientos funcionales y no funcionales de los servicios a ofrecer
2. Diseñar diagramas, modelos y arquitectura de los servicios a proponer
3. Desarrollar servicios de colas mensajes sobre operaciones para búsqueda de archivos
4. Realizar pruebas de los resultados obtenidos del desarrollo de los servicios.
5. Implementar el Buscador de Archivos desarrollado en el laboratorio de Sistemas Distribuidos de la Universidad Nacional Experimental de Guayana.

METODOLOGÍA DE DESARROLLO

Al momento de desarrollar sistemas informáticos es muy importante seleccionar la metodología apropiada para la exitosa elaboración del mismo; dependiendo del modelo seleccionado se puede mejorar la calidad del proyecto, aumentar la velocidad del desarrollo, minimizar los gastos y riesgos, entre otras ventajas.

En el desarrollo del Buscador de Archivos se empleó la metodología en cascada, esta es considerada como el enfoque clásico para el ciclo de vida del desarrollo de sistemas, se puede decir que es un método puro que implica un desarrollo rígido, es una secuencia de actividades (o etapas) que consisten en el análisis de requerimientos, diseño, codificación, pruebas e implementación, tal y como se indica en el siguiente gráfico.

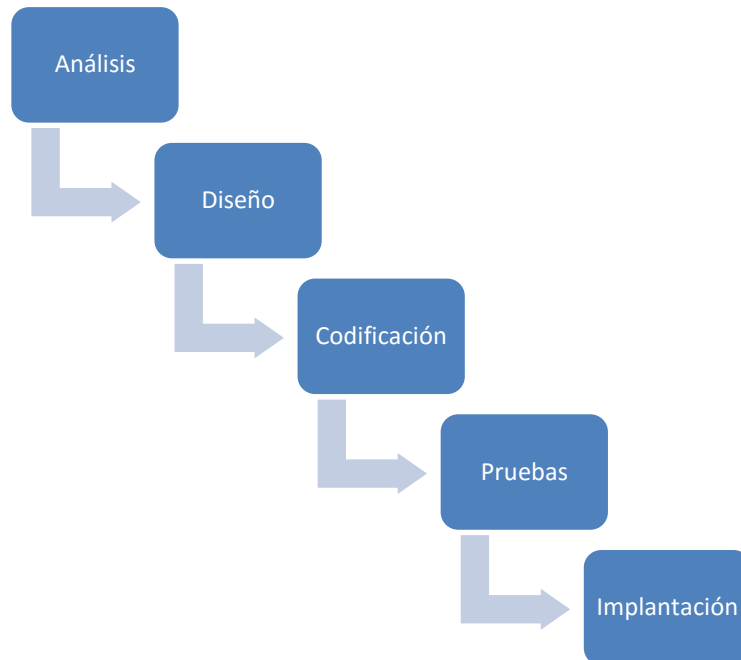


Ilustración 1 Metodología en Cascada

Etapas del modelo utilizado

Análisis de requerimientos

Se debe realizar un análisis tanto de los requerimientos funcionales como de informaciones necesarias y requeridas para las operaciones y funciones que ha de cumplir el sistema a desarrollar.

Requerimientos funcionales

1. Utilización de servicios de colas de mensajes que permita realizar búsquedas en equipos remotos, haciendo uso de librerías de python SnakeMQ.
2. Utilización de servicios web para la visualización y descarga de los archivos a través del navegador, apoyándose en librerías de flask de python.
3. Utilización del protocolo TCP/IP como método de transporte y modelo de fallas.
4. Modelado en UML.

Requerimientos no funcionales

1. Estudiar la arquitectura SnakeMQ.
2. Administrar correctamente las colas para el uso eficiente de mensajería dentro del Buscador de Archivos.
3. Garantizar la entrega de los mensajes con la dirección de los archivos basado en los parámetros de búsqueda cuando el cliente así lo solicite.

Diseño

Realizar los diseños de los diagramas pertinentes que para el desarrollo del sistema, así como también el diseño de la arquitectura en la que se basaran los servicios de colas de mensaje, siendo este el paso anterior a la parte de codificación, por lo tanto resulta necesario tener a disposición todos los diagramas y modelos en que estará basado el sistema.

Los diagramas desarrollados en UML se encuentran en el punto modelo de interacción, más específicamente las Ilustraciones 4, 5 y 6.

El diagrama de la arquitectura usada se encuentra en el punto arquitectura del sistema, más específicamente la Ilustración 2.

Codificación

Fase donde se inicia el desarrollo del sistema propuesta en base a los requerimientos ya establecidos en conjunto con el diseño del mismo que permita tener como base plasmada el funcionamiento que ha de tener el sistema.

Obtener IP de los equipos

Como se puede apreciar en la línea 66 en la Ilustración 2, existe un if para verificar en que sistema operativo se está corriendo el programa. Dicho programa es multiplataforma, es decir, puede ser corrido en Windows, y Linux solo en distribuciones basadas en Debian y Fedora.

```
62 def get_ip():
63     """
64     GetIP --> Funcion para obtener el ip
65     """
66     if platform.system().lower() == 'windows':
67         import socket
68         hostname = socket.gethostname()
69         ip = socket.gethostbyname(hostname)
70         return ip
71     else:
72         f = os.popen('ifconfig')
73         for iface in [' '.join(i) for i in iter(lambda: list(itertools.takewhile(lambda l: not l.isspace(), f)), [])]:
74             if re.findall('^(eth|wlan|wlp2s)[0-9]', iface) and re.findall('RUNNING', iface):
75                 ip = re.findall('(<=inet\saddr:)[0-9\.]+' , iface)
76                 if not ip:
77                     ip = re.findall('(<=inet\s)[0-9\.]+' , iface)
78                 if ip:
79                     return ip[0]
80     return False
```

Ilustración 2 Etapa Codificación, Función obtener IP equipos (servidor).

Servicio Web

```
40 # servicios web -----
41 @app.route('/<path:path>')
42 def static_file(path):
43     """
44     StaticFile --> Funcion para servir el archivo segun una ruta a traves de base 64
45     """
46     # decodifico el base64 para obtener la verdadera cadena que quiero procesar
47     filepath = str(base64.b64decode(path)).replace('\n', '')
48
49     # envio el archivo con la cadena decodificada
50     return send_file(filepath, attachment_filename=filepath[filepath.rfind('/'):])
51
52 def webserver():
53     """
54     WebServer --> Funcion para arrancar el servidor web
55     """
56     try:
57         # inicia servicios web, escuchando de todos lados, permitiendo conexiones remotas entrantes
58         app.run(host="0.0.0.0")
59     except:
60         pass
61 # -----
```

Ilustración 3 Etapa Codificación, Función para iniciar Servicio Web y función para servir archivo solicitado (servidor/slave)

```

165 @QtCore.pyqtSlot()
166 def browser_openfile(self):
167     # obtengo la fila seleccionada del grid al que le hice doble click
168     row = table_data.currentRow()
169
170     # abro navegador web con el host y direccion de archivo para visualizar o descargar, segun sea el caso
171     subprocess.call('google-chrome http://' + str(table_data.item(row, 0).text()) + ':5000/' + base64.b64encode(str(table_data.item(row, 1).text())) + '.<

```

Ilustración 4 Etapa Codificación, Función llamar al navegador con el archivo solicitado (cliente/master)

Permisología: Configuración carpetas de búsqueda

Recorre las líneas del archivo rootfolders.txt y lo asigna a un array que guarda las carpetas en memoria.

```

92 def get_root_folder_configuration():
93     """
94     GetRootFolderConfiguration --> Funcion para delvolver la ruta raiz de la carpeta donde se buscaran archivos
95     """
96     folders = []
97     try:
98         # abrir archivo en modo lectura
99         with open('rootfolders.txt', 'r') as f:
100             for line in f.readlines():
101                 line = line.replace('\n', '')
102                 # si linea leida del archivo de configuracion es una carpeta que existe en disco
103                 if os.path.isdir(line):
104                     # agregarlo al path de carpetas permitidas para buscar
105                     folders.append(line)
106
107     except:
108         pass
109
110     # si no hay ninguna carpeta valida, agregar el root /home
111     if folders.__len__() == 0:
112         folders.append('/home')
113
114     # en caso de error, retornar raiz
115     return folders

```

Ilustración 5 Etapa Codificación, Función permisología configuración de carpetas (servidor)

Buscar archivos

Función que realiza la tarea de la búsqueda de los archivos desde las carpetas indicadas en el archivo rootfolders.txt, y notifica a los demás nodos en los otros niveles que se está haciendo una solicitud de una búsqueda.

```
118 def search(search_data):
119     """
120     Search --> Funcion para buscar recursivamente un archivo en el computador
121     """
122     global m
123     global host
124     global slaves
125     global master
126
127     # enviar mensaje a todos los nodos que se identificaron, para buscar de manera recursiva en todos los nodos
128     send_messages_to_idents_hosts(search_data)
129     print("Buscando --> " + search_data + " ")
130
131     # buscar archivo en las carpetas asignadas en la configuracion
132     for root_folder in search_folders:
133         # buscar archivos de manera recursiva, a partir de la raiz de esta carpeta
134         for root, dirnames, filenames in os.walk(root_folder):
135             # si el archivo cumple con el patron de busqueda
136             for filename in fnmatch.filter(filenames, search_data):
137                 # construyo mensaje, indicando host y direccion de archivo encontrado
138                 msg = snkemq.message.Message(str({
139                     'host': host,
140                     'data': root + '/' + filename
141                 }).encode(), ttl=60)
142                 # si tengo el MASTER conectado a mi, le notifico a el
143                 if master:
144                     m.send_message("master", msg)
145                 else:
146                     # en caso contrario, replico la respuesta a los nodos servidores, para hacerle llegar la informacion al MASTER
147                     send_messages_to_servers_hosts(msg)
148                 print("Notificando --> " + root + filename)
```

Ilustración 6 Etapa Codificación, Función Buscar Archivos (servidor)

Ventana Principal Cliente (Master)

```
94 def initUI(self):
95     """
96     Initialize User Interface --> Funcion instanciar la ventana principal
97     """
98     self.resize(640, 480)
99     self.setWindowTitle("SD_Garcia_Etcheverry - Master")
100    self.center()
101
102    label_search = QLabel("<h4>Archivo:</h4>", self)
103    label_search.move(30, 30)
104
105    self.line_search = QLineEdit("", self)
106    self.line_search.setGeometry(100, 26, 200, 25)
107
108    button_search = QPushButton("Buscar", self)
109    button_search.move(310, 25)
110    button_search.clicked.connect(self.search)
111
112    button_clear = QPushButton("Limpiar", self)
113    button_clear.move(410, 25)
114    button_clear.clicked.connect(self.clear)
115
116    button_conf = QPushButton("Configuracion", self)
117    button_conf.move(510, 25)
118    button_conf.clicked.connect(self.configuration)
119
120    global table_data
121    table_data = QTableWidget(0, 2, self)
122    table_data.setHorizontalHeaderLabels(("Host", "Direccion"))
123    table_data.setGeometry(30, 75, 580, 380)
124    table_data.setColumnWidth(0, 120)
125    table_data.setColumnWidth(1, 419)
126    table_data.doubleClicked.connect(self.browser_openfile)
127
128    self.configuration_window = ConfigWindow.ConfigWindow()
```

Ilustración 7 Etapa Codificación, Función Ventana Principal Cliente

Ventana Configuración Cliente (Master)

```
23 def initUI(self):
24     """
25     Initialize User Interface --> Funcion instanciar la ventana principal
26     """
27     self.resize(320, 240)
28     self.setWindowTitle("Configuracion")
29     self.center()
30
31     label_description = QLabel("<h4>Host Esclavos:</h4>", self)
32     label_description.move(20, 15)
33
34     self.text_edit_host = QTextEdit(self)
35     self.text_edit_host.setGeometry(20, 40, 280, 180)
36     with open('host.txt', 'r') as f:
37         for line in f.readlines():
38             self.text_edit_host.append(line.replace('\n', ''))
39
40     button_save = QPushButton("Guardar", self)
41     button_save.move(220, 10)
42     button_save.clicked.connect(self.hosts_save)
```

Ilustración 8 Etapa Codificación, Función Ventana Configuración Cliente

Guardar los host configurados (Cliente)

```
53 def hosts_save(self):
54     """
55     HostsSave --> Funcion para guardar los hosts configurados
56     """
57     # abro el archivo de hosts en modo escritura y escribo en el archivo lo que configure
58     with open('host.txt', 'w') as f:
59         f.write(self.text_edit_host.toPlainText())
60
61     # muestro mensaje de que morira la aplicacion y debera iniciarla nuevamente
62     QMessageBox.about(self, "Configuracion", "Guardada exitosamente!\nInicie la aplicacion nuevamente")
63
64     # mato la aplicacion con signal kill forced
65     subprocess.call('kill -9 ' + str(os.getpid()), shell=True)
```

Ilustración 9 Etapa Codificación, Función guardar host configurados (Cliente)

Archivo encontrado por los servidores, en la vista cliente

```
31 def on_recv(conn, ident, message):
32     """
33     OnRecv --> Funcion disparadora al llegar un mensaje
34     """
35     # hago un cast del string del mensaje llegado a un json
36     msg = json.loads(str(message.data).replace('\n', ''))
37
38     # obtengo cual va a ser la posicion de la ultima fila
39     rowPosition = table_data.rowCount()
40
41     # inserto una fila en el grid
42     table_data.insertRow(rowPosition)
43
44     # asigno los valores devueltos de la busqueda a su posiciones correspondientes, HOST y FILE FOUND
45     table_data.setItem(rowPosition, 0, QTableWidgetItem(msg['host']))
46     table_data.setItem(rowPosition, 1, QTableWidgetItem(msg['data']))
47
48
```

Ilustración 10 Etapa Codificación, Función archivo encontrado (Cliente)

Pruebas

En esta etapa se verifica que el comportamiento del sistema desarrollado cumple con los requerimientos establecidos con anterioridad en las fases iniciales, cumpliendo con el modelado para el sistema, de modo que sirve como comparación de calidad para establecer el funcionamiento de las operaciones que ha de cumplir el sistema

Dichas pruebas se encuentran en el punto pruebas y resultados.

Implementación

Una vez desarrollado el sistema y luego de haberle realizado las pruebas pertinentes, este ha de ser implementado en el lugar donde sea requerido, para ser usado por los usuarios finales destinado a darle provecho a las funcionalidades establecidas y alcanzadas.

HARDWARE UTILIZADO

Se utilizan específicamente tres (3) computadoras del Laboratorio de Avanzada de la Universidad Nacional Experimental de Guayana, aunque la cantidad mínima de equipos necesaria para correr dicho proyecto es dos (2).

Especificaciones de las computadoras:

Detalles	Equipo 1	Equipo 2	Equipo 3
Memoria	489,7 MiB	999,4 MiB	997,3 MiB
Procesador	Intel Pentium® 4 CPU 3.2 GHz x 2	Intel Pentium® 4 CPU 2.80 GHz x 2	Intel Pentium® 4 CPU 3.2 GHz x 2
Gráficos	Intel ® 9456 x86/MMX/SSE2	Intel ® 9456 x86/MMX/SSE2	Intel ® 9456 x86/MMX/SSE2
Sistema base	Debian GNU/Linux 8 (jessie) 32-bit	Debian GNU/Linux 8 (jessie) 32-bit	Debian GNU/Linux 8 (jessie) 32-bit
Disco	77,6 GB	76,5 GB	76,5 GB

Para el correcto funcionamiento del proyecto se instalaron en cada una de las computadoras los siguientes programas y librerías:

1. Librerías de qt para python.
2. Pip para python.
3. Snakemq para python.
4. Flask para python.

Arquitectura del sistema

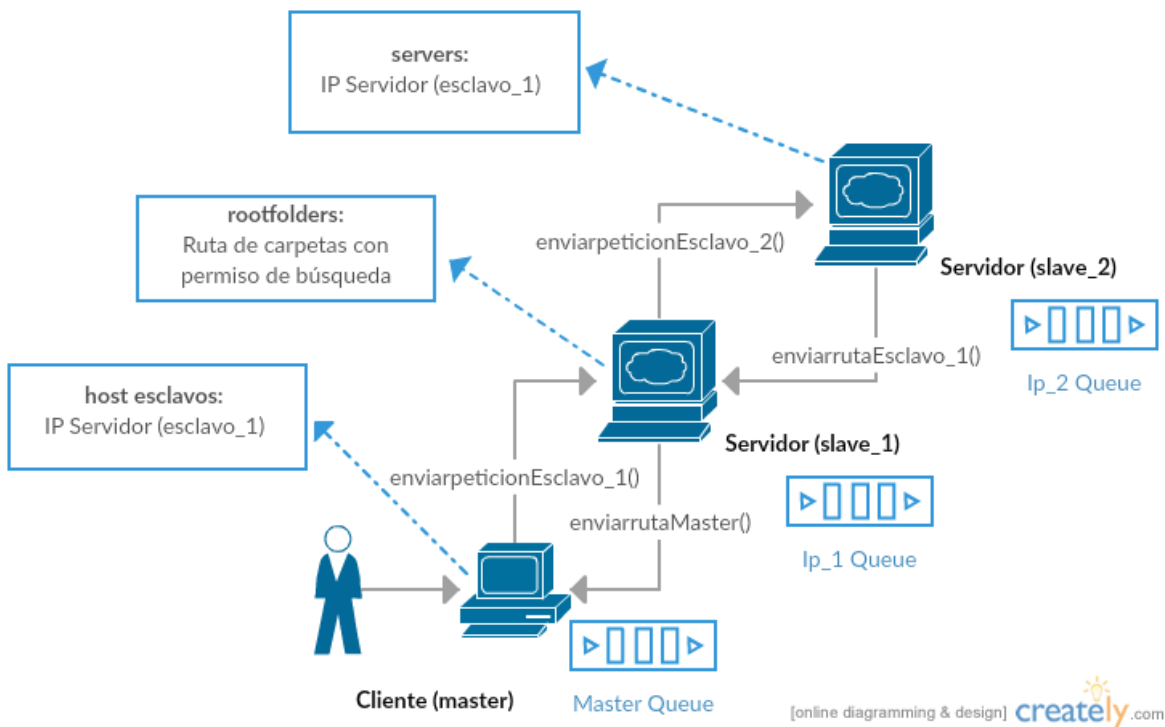


Ilustración 11 Diagrama Arquitectura del Sistema

Dónde:

- Cliente (master): Nodo del cliente, quién realizará las búsquedas en el sistema.
- Servidor (slave_1): Nodo del servidor que hace función de puente entre el cliente y demás nodos.
- Servidor (slave_2): Nodo servidor.

Cada nodo dentro del sistema posee una cola de mensajes, dando un total de tres colas.

Cada cola de mensaje posee como identificador su IP para el caso de los servidores, y “master” en el caso de la cola del cliente.

Estructura de las colas de mensajes



Ilustración 12 Estructura cola de mensaje

Modelo de interacción

El proyecto basa su estructura en el Lenguaje Unificado de Modelado (UML), el cual para el presente proyecto ayuda a describir de una forma eficiente métodos y/o modelos, además, es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema.

A continuación se mostrarán los diagramas de caso de uso y secuencia.

Diagrama de Caso de uso

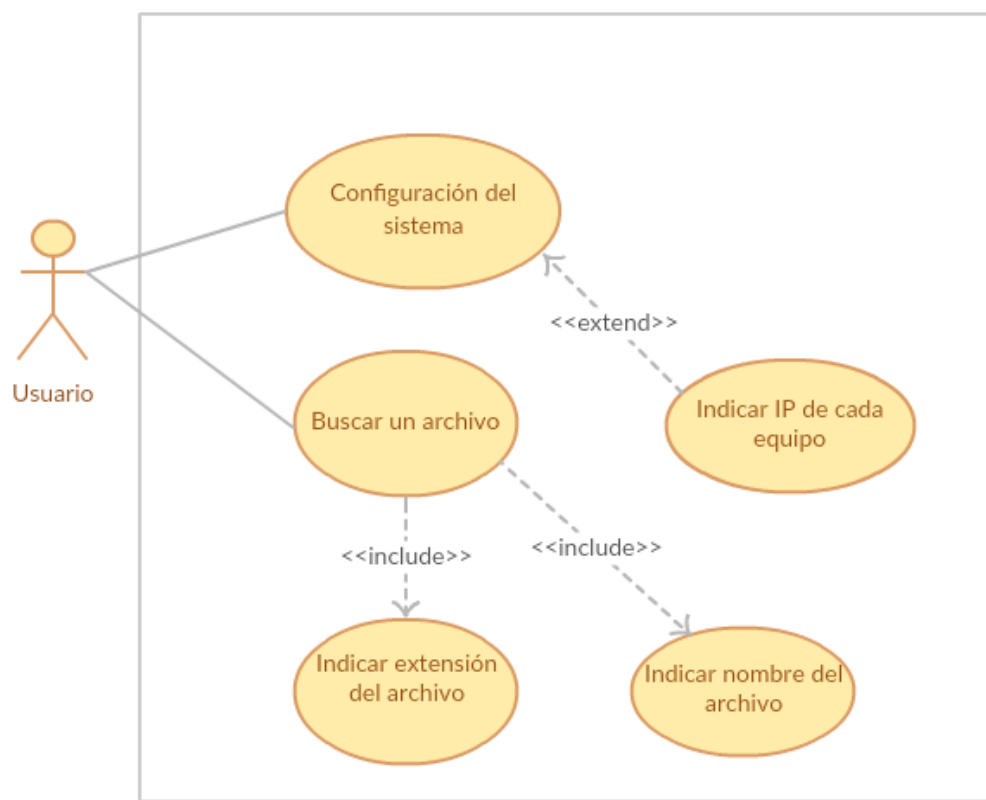


Ilustración 13 Diagrama Caso de Uso

Diagrama de secuencia configuración inicial, creación del sistema de colas.

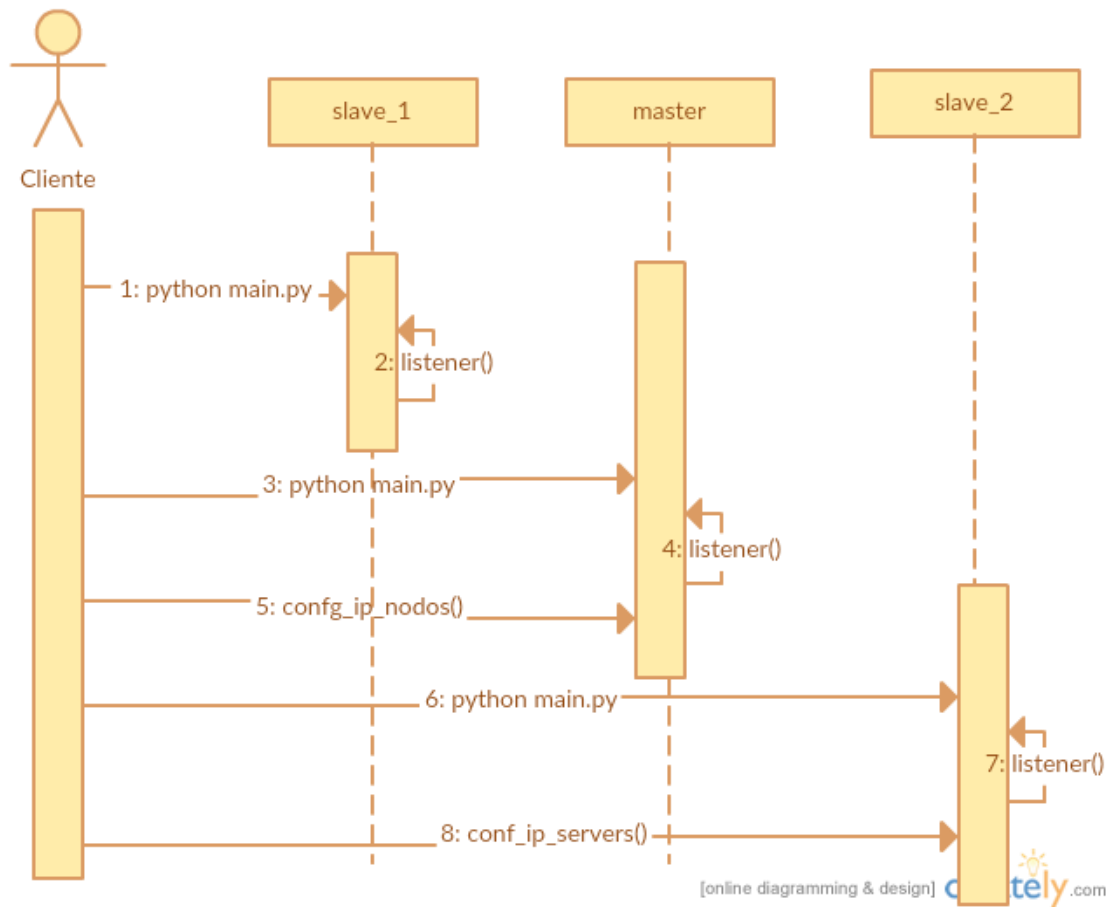


Ilustración 14 Diagrama de secuencia para configuración inicial

Diagrama de secuencia búsqueda de archivos.

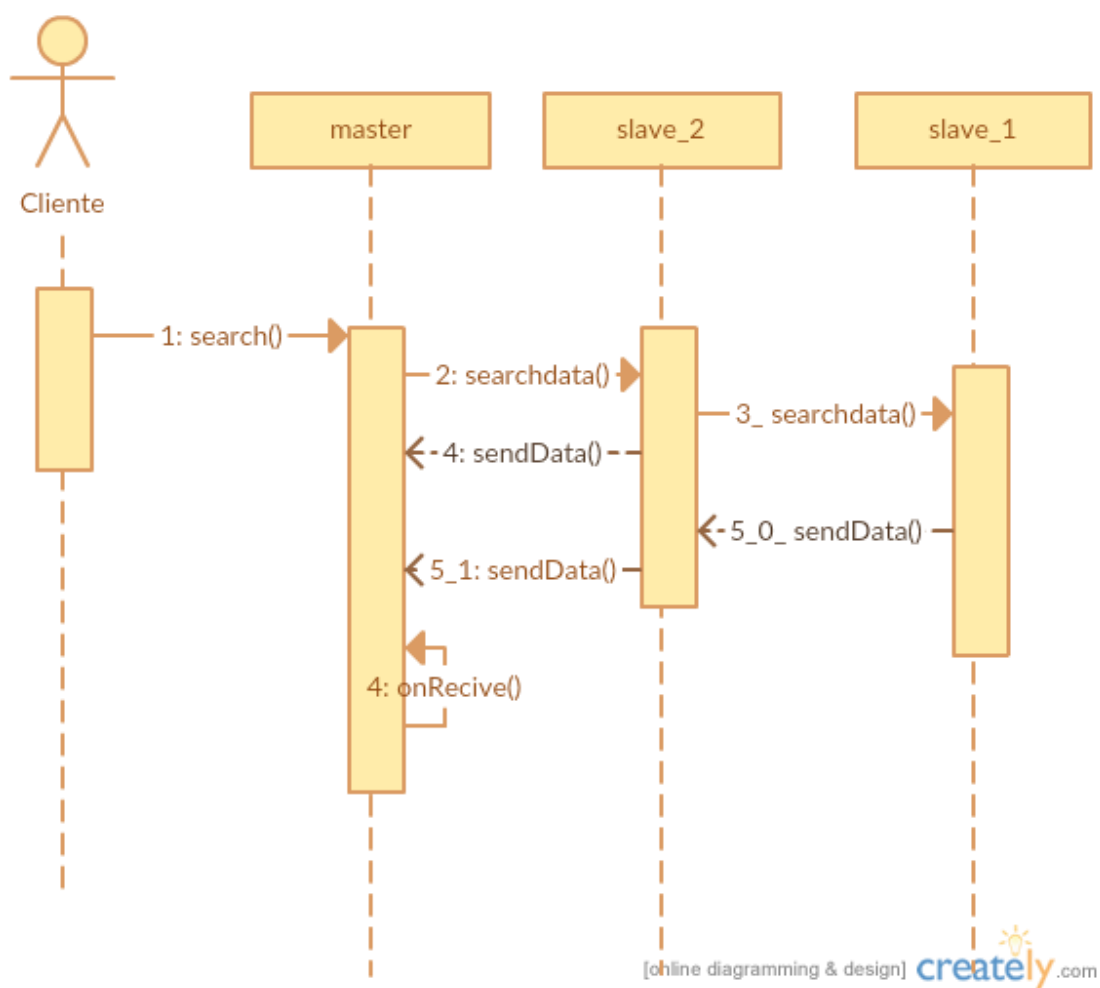
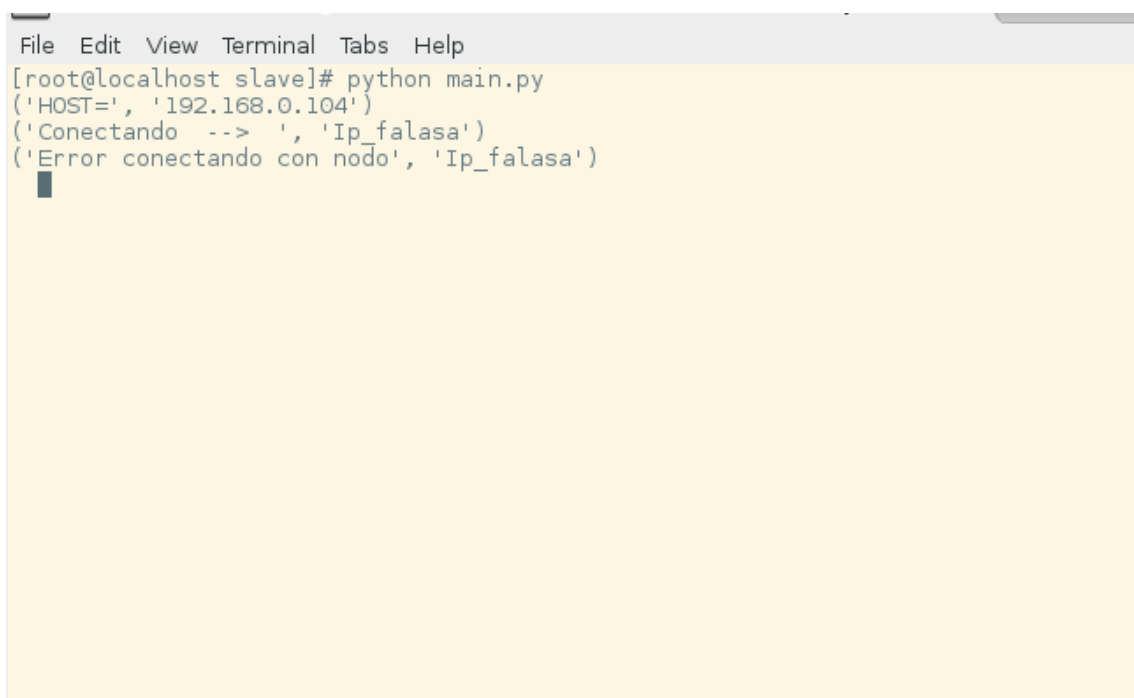


Ilustración 15 Diagrama de secuencia para la búsqueda de archivos

Modelo de fallas

Error de IP

El archivo servers.txt posee las IP del master y slave2, para el presente caso de prueba se introdujo en el archivo “Ip_falasa” en lugar de una IP real, al momento de correr la aplicación se generará el error de conexión a nodo, imprimiendo en pantalla el mensaje de error correspondiente.

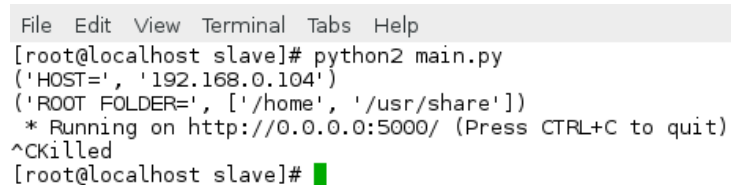
A screenshot of a terminal window with a menu bar (File, Edit, View, Terminal, Tabs, Help) and a yellow background. The terminal shows the execution of a Python script. The output includes the host IP '192.168.0.104', a connection attempt to 'Ip_falasa', and an error message: 'Error conectando con nodo', 'Ip_falasa'.

```
File Edit View Terminal Tabs Help
[root@localhost slave]# python main.py
('HOST=', '192.168.0.104')
('Conectando --> ', 'Ip_falasa')
('Error conectando con nodo', 'Ip_falasa')
```

Ilustración 16 Modelo de fallas: Error de IP

Eliminación de nodos

Para la eliminación de nodos durante la ejecución se puede ejecutar el comando: Ctrl+C, esto termina (mata) el proceso.

A screenshot of a terminal window with a menu bar (File, Edit, View, Terminal, Tabs, Help) and a white background. The terminal shows the execution of a Python script. The output includes the host IP '192.168.0.104', the root folder path, and a message indicating the application is running on http://0.0.0.0:5000/. The process is then terminated with Ctrl+C, resulting in a '^CKilled' message.

```
File Edit View Terminal Tabs Help
[root@localhost slave]# python2 main.py
('HOST=', '192.168.0.104')
('ROOT FOLDER=', ['/home', '/usr/share'])
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
^CKilled
[root@localhost slave]#
```

Ilustración 17 Modelo de fallas: Eliminación de nodos

Detección de nodos: Proceso de incorporación y desincorporación de nodos

Aquellos nodos que hacen función de puentes dentro del sistema, realizan la detección de la conexión de otros servidores y masters (incorporación), a su vez, también realizan la detección en el caso contrario, es decir, cuando dichos nodos se desconectan (desincorporación). Esto con la intención de poder detectar qué nodos hay en la red y si alguno de ellos se cae durante la ejecución del sistema.

```
File Edit View Terminal Tabs Help
[root@localhost slave]# python2 main.py
('HOST=', '192.168.0.104')
('ROOT FOLDER=', ['/home', '/usr/share'])
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
('Se ha conectado un nodo de busqueda --> ', u'192.168.0.100')
Se ha establecido conexion con el nodo MASTER
('Desconectado nodo ', '1fd8')
Desconectado nodo MASTER
```

Ilustración 18 Modelo de fallas: Proceso de incorporación y desincorporación de nodos

Instalación

A continuación se presentan los comandos que se ejecutaron para ambientar las maquinas con las tecnologías requeridas para la correcta ejecución del proyecto.

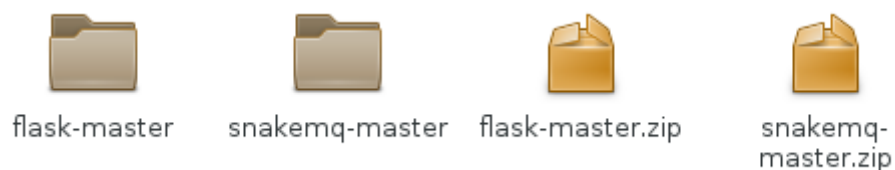
COMANDO	DESCRIPCIÓN
apt-get install python-qt4	Librerías de qt para python.
apt-get install aptitude	Manejador de paquetes de linux debían.
aptitude update	Actualizar repositorio local.
apt-get install python-pip	Instalar pip para python.
pip install snakemq	Instalar paquete de snakemq para python.
pip install flask	Instalar librerías del flask para correr servicios web

En caso de no tener internet existen estos archivos que han sido descargados de la página fuente y que pueden ser ejecutados manualmente, a continuación los pasos:

1. Copiar los siguientes comprimidos en el equipo



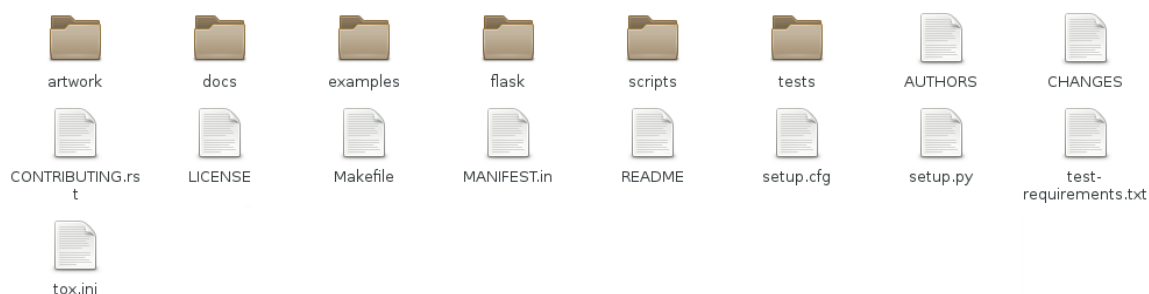
2. Extraer archivos comprimidos



3. Ejecutar en la terminal de cada una de las carpetas, en modo root.

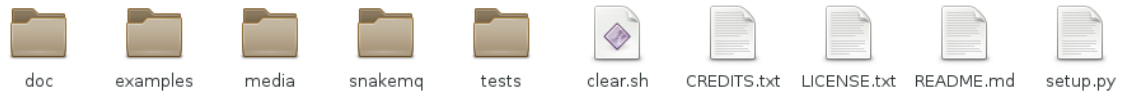
- a. En la carpeta flask-master

```
File Edit View Terminal Tabs Help
[root@localhost flask-master]# python2 setup.py install
```



- b. En la carpeta snakemq-master

```
File Edit View Terminal Tabs Help
[root@localhost Downloads]# cd snakemq-master
[root@localhost snakemq-master]# python2 setup.py install
```



4. Verificación de instalación de librerías

```
File Edit View Terminal Tabs Help
[root@localhost ~]# pip2 list | grep snakeMQ
snakeMQ (1.2)
[root@localhost ~]# pip2 list | grep Flask
Flask (0.11.1)
[root@localhost ~]#
```

Pasos para configurar el Buscador de archivos:

1. Permisología: En este paso se indican las carpetas en las que el usuario puede buscar dentro de los servidores. Esto restringe las búsquedas del usuario, por razones de seguridad evitando que el usuario pueda obtener archivos importantes del sistema.

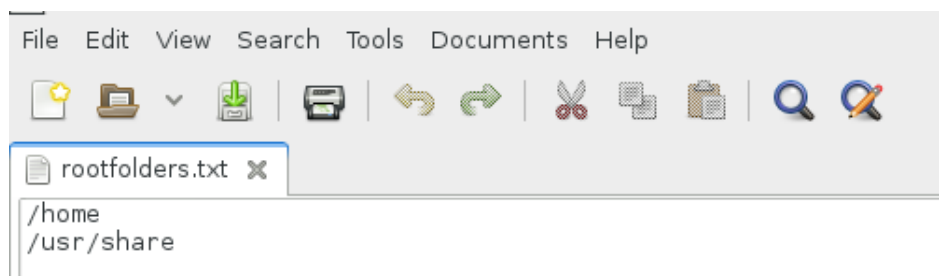


Ilustración 19 Instalación, Configuración de permisología

2. Configuración host en Cliente (Master): Se incluyen los IP de aquellos servers en el archivo host.txt.

Tal y como se muestra en la Ilustración 10 a continuación, es una ventana que contiene las direcciones IP de los Servidores Host Esclavos.



Ilustración 20 Instalación, Configuración Host desde el Cliente

3. Configuración Servidor_2 (Slave): El nodo que cumple la función de servidor_2, tal y como se explica en el diagrama expuesto en el punto arquitectura del sistema (Ilustración 2), este nodo se configura en un nivel que no es visto por el nodo cliente, por lo que para establecer conexión con él, se configura en el archivo servers.txt el IP del nodo que realizará la función de puente entre este y la red de nodos que llegarán al cliente.

La Ilustración 11 a continuación expone los archivos necesarios para la ejecución del servidor_1 (Slave_2), indicando a su vez la configuración que se realiza a través servers.txt del archivo previamente a la ejecución.

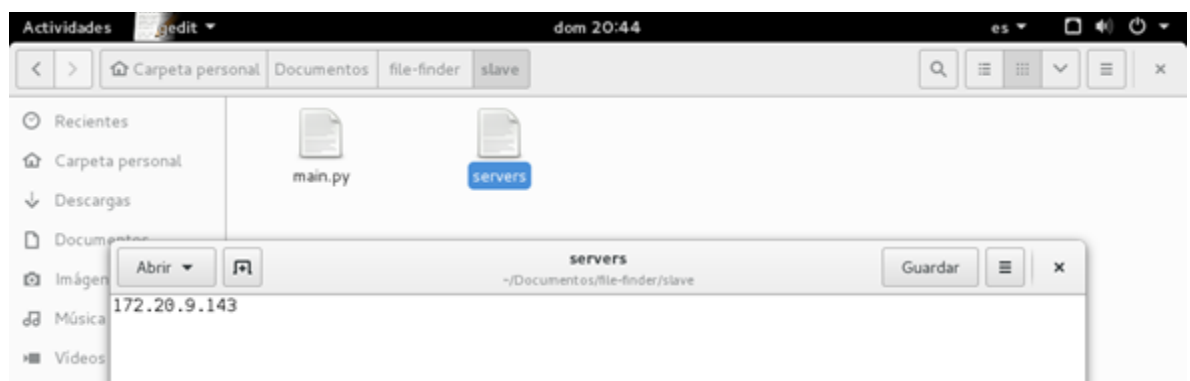


Ilustración 21 Instalación, Configuración Servidor_2 (Slave)

4. Servidor_1 (Slave_1): Tal y como se explica en el diagrama expuesto en el punto arquitectura del sistema en la Ilustración 2, el servidor_1 (slave_1) hace la función de un nodo tipo “puente” entre los nodos: cliente (master) y servidor_2 (slave_2), es decir, este nodo permite la comunicación y propagación de órdenes de búsquedas de archivos en otros nodos que el nodo cliente desconoce.



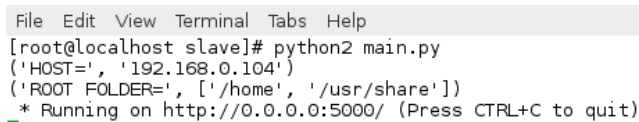
Ilustración 22 Instalació, Servidor_1 (Slave)

5. Se procede a realizar las ejecuciones de los distintos programas:

Para la realización de los siguientes pasos es necesario estar en modo root (super usuario), esto es debido a que el programa ejecuta el comando ifconfig para conocer las IP de cada equipo, recordando que solo se puede ejecutar ifconfig bajo el modo root.

Tal y como se explica en la Ilustración 2 que expone las líneas de código que realizan la tarea de obtener IP.

- Ejecución Cliente (Master): Se procede a correr la aplicación de cliente con el comando `python main.py`
- Ejecución Servidores (Esclavos): Se proceder a correr las aplicaciones de los servidores con el comando `python main.py`



```
File Edit View Terminal Tabs Help
[root@localhost slave]# python2 main.py
('HOST=', '192.168.0.104')
('ROOT_FOLDER=', ['/home', '/usr/share'])
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Ilustración 23 Instalación, Comando para ejecución de servidores y cliente

Características comunes para servidor_1 y servidor_2:

- Ambos nodos al ser ejecutados, imprimen en pantalla su IP, la cual a su vez sirve de nombre para identificar su cola de mensajes correspondiente dentro del sistema.
- Cuando el cliente inicia una búsqueda los esclavos indican qué archivo están buscando, más específicamente: el nombre con su respectiva extensión.
- En cada ocurrencia que los servidores registren, enviarán una notificación al cliente (master).

Búsqueda de archivos desde el cliente (master)

En la Ilustración 14 se muestra el comando para ejecutar el programa del maestro `python main.py`, dicho comando levanta la ventana del cliente la cual contiene las siguientes opciones:

- Input “Archivo”:** Es aquí donde el usuario introduce el nombre del archivo a buscar con su respectiva extensión
- Botón “Buscar”:** Opción que inicia el proceso de búsqueda a los nodos servidores (slave) que se encuentran configurados.
- Botón “Limpiar”:** Opción que limpia la lista que se encuentra en la parte inferior

- **Botón “Configuración”**: Opción que permite al cliente configurar aquellos nodos en los que desea realizar la búsqueda, esta ventana se puede apreciar en la Ilustración 10.

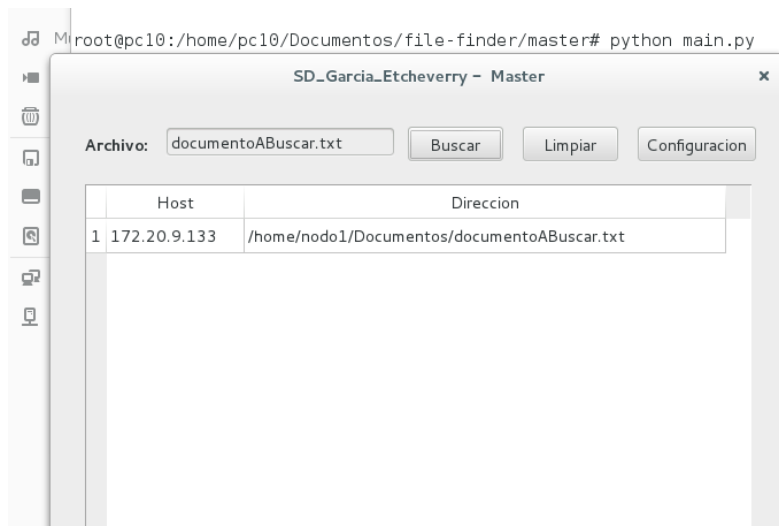


Ilustración 24 Instalación, Ventana Cliente (Master)

PRUEBAS Y RESULTADOS

Búsqueda de un archivo basado en su extensión

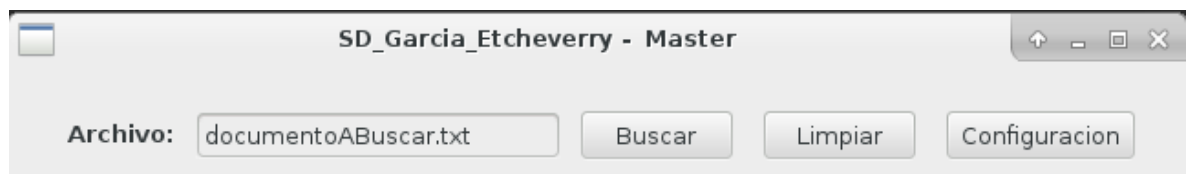
A continuación se evidencia la búsqueda de todo archivo *.c, es decir todo archivo que su extensión sea .c.



Ilustración 25 Prueba Servidor_2, Búsqueda archivo *.c

Búsqueda de un archivo basado en su nombre y extensión

Se comienza una nueva búsqueda, del archivo documentoABuscar.txt, el servidor que posee el archivo envía una notificación al cliente indicando la dirección del mismo dentro del disco.



Ubicación del archivo

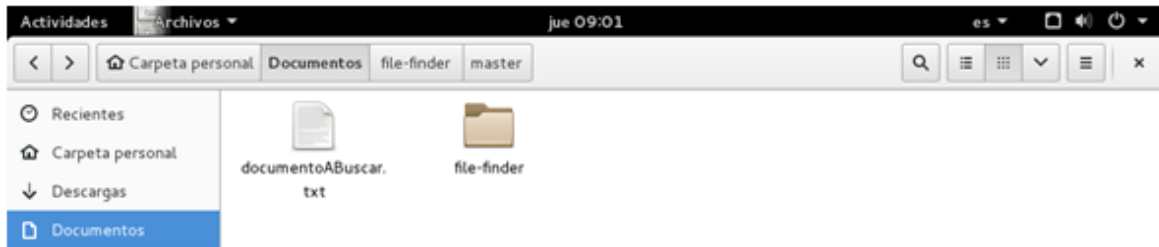


Ilustración 26 Prueba, ubicación archivo a buscar documentoABuscar.txt

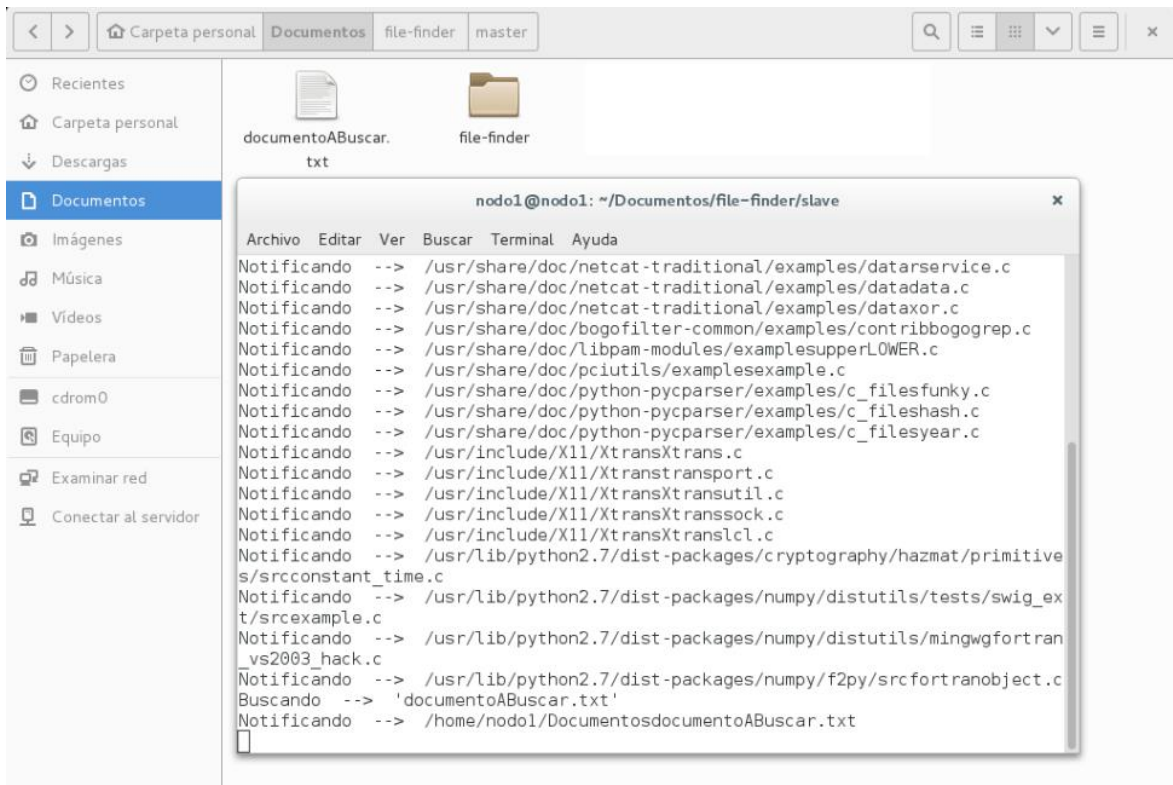


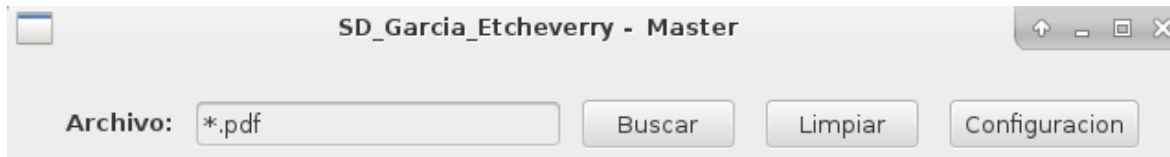
Ilustración 27 Prueba notificación de la ruta del archivo documentoABuscar.txt por parte del Servidor_2

Búsqueda de dos archivos simultáneamente (conurrencia) desde un mismo cliente

Paso 1: Se realiza una búsqueda de archivos *.c



Paso 2: Se realiza una búsqueda de archivos *.pdf



Paso 2: El servidor recibe ambas peticiones de búsqueda

```
File Edit View Terminal Tabs Help
[root@localhost slave]# python2 main.py
('HOST=', '192.168.0.104')
('ROOT FOLDER=', ['/home', '/usr/share'])
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
Se ha establecido conexion con el nodo MASTER
Buscando --> '*.c'
Buscando --> '*.pdf'
```

Ilustración 28 Prueba de varias peticiones simultáneamente (concurrency)

Paso 2: El servidor envía las notificaciones como respuestas a ambas peticiones de búsqueda

```
File Edit View Terminal Tabs Help
[root@localhost slave]# python2 main.py
('HOST=', '192.168.0.104')
('ROOT FOLDER=', ['/home', '/usr/share'])
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
Se ha establecido conexion con el nodo MASTER
Buscando --> '*.c'
Buscando --> '*.pdf'
Notificando --> /usr/share/doc/libpng-1.5.13example.c
Notificando --> /usr/share/doc/libtasn1-3.8libtasn1.pdf
Notificando --> /usr/share/doc/systemdsd-readahead.c
Notificando --> /usr/share/doc/ppp-2.4.5/scripts/chatchatchatchat.c
Notificando --> /usr/share/doc/aic94xx-firmware-30README-94xx.pdf
Notificando --> /usr/share/doc/libbluray-0.2.3/player_wrappers/xineinput_bluray.c
Notificando --> /usr/share/doc/mpfr-3.1.1/mpfr/examplesdivworst.c
Notificando --> /usr/share/doc/mpfr-3.1.1/mpfr/examplesrndo-add.c
Notificando --> /usr/share/doc/mpfr-3.1.1/mpfr/examplessample.c
Notificando --> /usr/share/doc/mpfr-3.1.1/mpfr/examplesversion.c
Notificando --> /usr/share/doc/libao-1.1.0ao_example.c
```

Ilustración 29 Prueba, notificación de archivos en peticiones simultaneas (servidor)

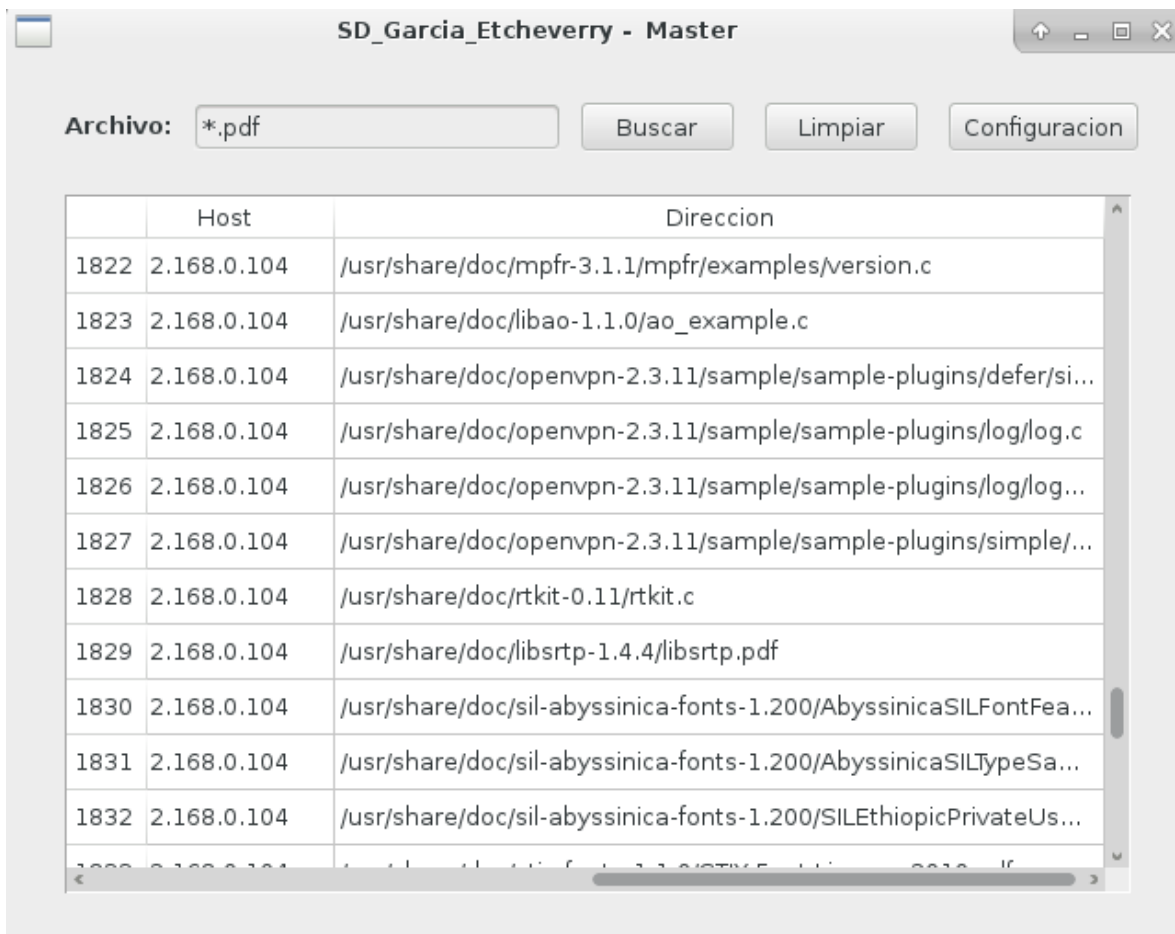


Ilustración 30 Prueba Ventana Cliente, resultado de peticiones simultáneamente.

Visualización de un archivo a través del navegador

Luego de que el cliente realice la búsqueda de un archivo, tiene la opción de visualizarlo a través del navegador, haciendo doble click en el ítem dentro de la lista de rutas desde la ventana, para ello el programa sigue los siguientes parámetros:

- De ser un archivo entendible por el navegador este se mostrará, ejemplo: un archivo pdf.
- De no ser un archivo entendible por el navegador este dará la opción de poder descargarlo, ejemplo: un archivo .c

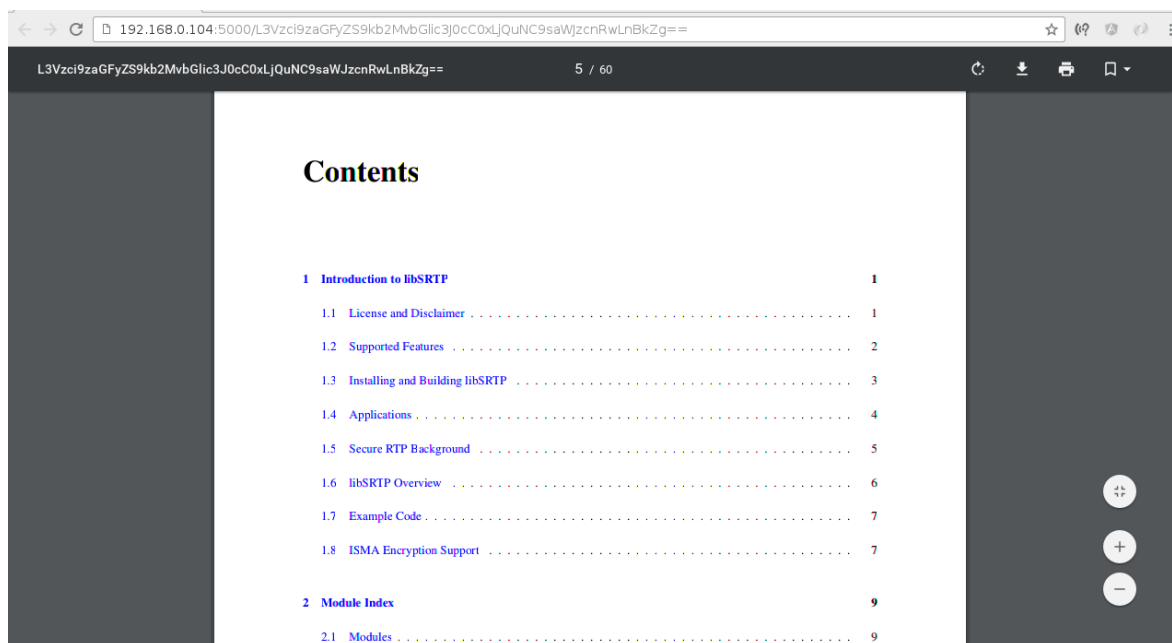
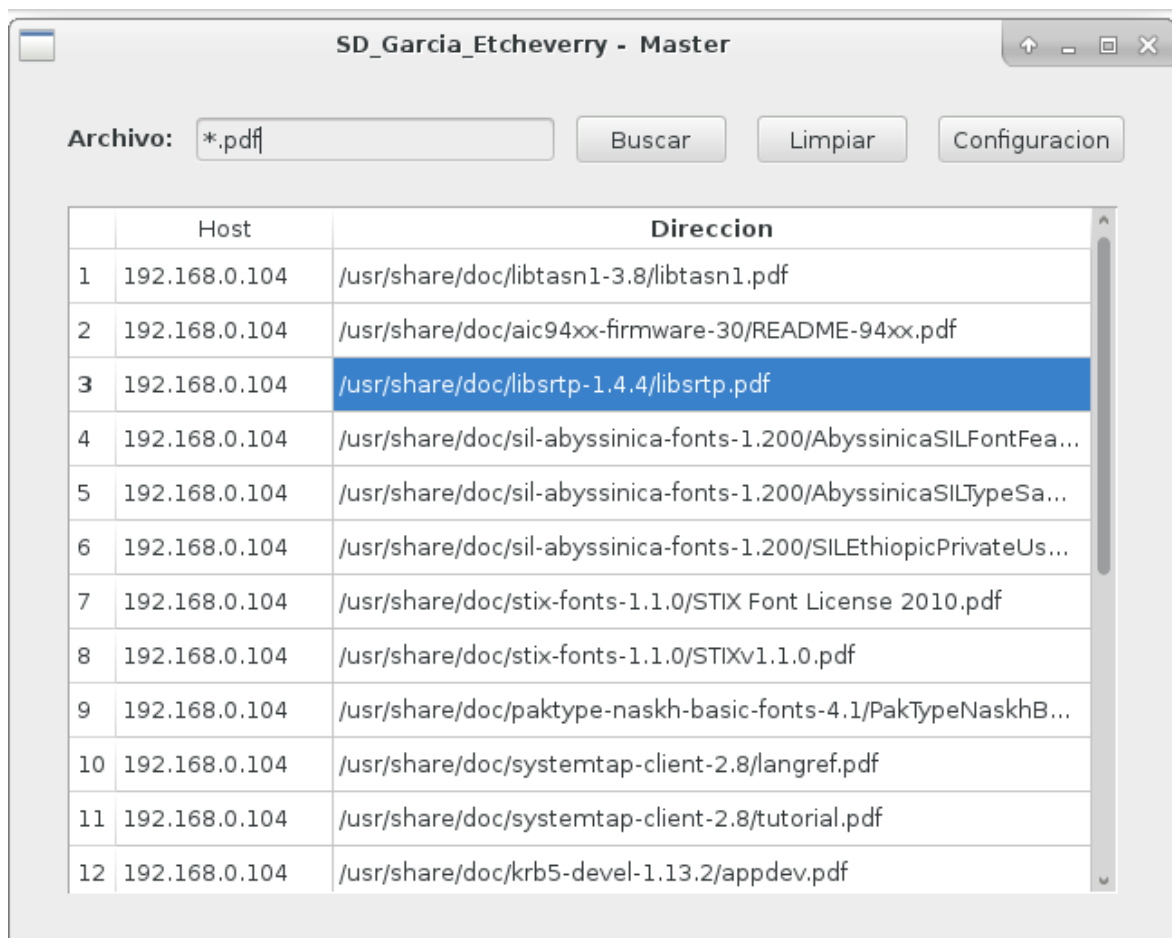


Ilustración 31 Prueba, visualización de un archivo pdf en el navegador del cliente

CONCLUSIÓN

Los requerimientos funcionales y no funcionales fueron analizados correctamente y se logró detallar las necesidades del Buscador de Archivos, concluyendo que este tiene como tarea la administración de colas para resolver las peticiones del usuario dependiendo de sus parámetros de búsqueda, donde se tiene como principal requisito que el usuario provea toda la información necesaria para una búsqueda eficiente.

Se estudió la documentación SnakeMQ para comprender como trabaja la arquitectura de colas de mensajes, con el fin de obtener un correcto manejo del sistema. La arquitectura se utilizó para dar soporte a través de colas de mensaje en la entrega de peticiones por parte del cliente y notificación de respuestas por parte de los servidores, permitiendo al Buscador de Archivos gestionar cada petición en el orden correspondiente, el primero en llegar es el primero en salir (ser atendido).

El diseño del Buscador de archivos, cumple con las características de simplicidad, fácil manejo, escalabilidad y multiplataforma. El modelo de fallas fue diseñado para soportar los errores de configuración de IP, para el caso de introducir una IP falsa o con errores, restricción de búsquedas más específicamente manejo de permisología para control de seguridad, así como también el manejo de incorporación y desincorporación de nodos durante la ejecución del mismo.

El modelado del proyecto basa su estructura en el Lenguaje Unificado de Modelado (UML), el cual ayudó a describir de una forma eficiente el comportamiento, además, es un lenguaje gráfico para visualizar y documentar el sistema.

Se desarrolló el Buscador de Archivos con la arquitectura de colas de mensajes SnakeMQ, apoyándose en el lenguaje python y sus correspondientes librerías de SnakeMQ, complementando con librerías Flask para la utilización del navegador como método de visualización y descarga de archivos encontrados en los servidores.

Con las pruebas realizadas, y rectificando el correcto funcionamiento del Buscador de Archivos, se puede implementar para la Universidad Nacional Experimental de Guayana, como herramienta de búsqueda de archivos dentro de los laboratorios, biblioteca y otras áreas de interés.

BIBLIOGRAFÍA

[MICROSOFT (2016)]. *Información básica sobre colas de mensajes y tecnología de mensajería*.

Página Web en Línea. Disponible en:

[https://msdn.microsoft.com/es-es/library/19ww660c\(v=vs.90\).aspx](https://msdn.microsoft.com/es-es/library/19ww660c(v=vs.90).aspx) (Consultado el 23 de Junio del 2016).

[SNAKEMQ (2015)]. *SnakeMQ - message queuing for Python*. Página Web en Línea. Disponible en:

<http://www.snakemq.net/> (Consultado el 23 de Junio del 2016).