



**Instituto Politécnico Nacional
Escuela Superior de Cómputo
Academia de Ingeniería de Software**

Práctica 2

Integrantes del equipo:

**Castro Flores Marcela
Sánchez Cruz Rosa María
Santiago Mancera Arturo Samuel**

M. en C. Tanibet Pérez de los Santos Mondragón

México, Ciudad de México a 04 de noviembre de 2018

Índice general

1. Introducción	4
2. Algoritmos implementados	5
2.1. Algoritmo de Línea de Base	5
2.1.1. Explicación de código	8
2.2. Algoritmo de Mínimos Cuadrados	10
2.3. Algoritmo de Holt Winters	17
2.3.1. Explicación de código	21
3. Cuestionario	27
4. Conclusiones	28

Índice de figuras

2.1. Inventario.	5
2.2. Umbrales de host.	6
2.3. Correo 1.	7
2.4. Correo 2.	7
2.5. Correo 3.	8
2.6. Código para generar gráficas.	8
2.7. Verificación de alertas	9
2.8. Llamada a notificar	9
2.9. Envío de correo.	9
2.10. Inicio de la interfaz de mínimos cuadrados.	10
2.11. Ingreso de valores.	11
2.12. Validación de campos de datos.	12
2.13. Función graficar.	13
2.14. Función checkCPU.	14
2.15. Validación de la función checkCPU.	14
2.16. Función graficar.	15
2.17. Valores para iniciar la grafica de Mínimos cuadrados.	16
2.18. Gráfica resultante del algoritmo de mínimos cuadrados.	16
2.19. Nuevo agente añadido.	17
2.20. Pantalla principal de agentes.	18
2.21. Información del agente activo.	18
2.22. Ventana de gráficos disponibles.	18
2.23. Gráfica de OutNUCastPkts.	18
2.24. Gráfica de OutNUCastPkts.	19
2.25. Gráfica de InOctets.	19
2.26. Correo de inicio del error.	20
2.27. Correo de fin del error.	20
2.28. Correo de inicio del error.	21
2.29. Correo de fin del error.	21
2.30. Método actualizarHW.	22
2.31. Método crearHW.	22
2.32. Método actualizarHW.	23
2.33. Método graficar_HW.	24
2.34. Método check_aberration.	25
2.35. Método send_alert_attached.	26

CAPÍTULO 1

Introducción

Para la realización de esta práctica se utilizó nuevamente el protocolo SNMP, sin embargo esta vez se implementaron 3 algoritmos que serán explicados posteriormente.

Dichos algoritmos son:

- Línea de base
- Mínimos cuadrados
- Holt Winters

Esta práctica se dividió en las tres partes siguientes:

1. En la primera parte de este documento se aborda el tema de Línea base, el cual es un mecanismo para definir umbrales ante los niveles reportados por un agente dependiendo de las muestras enviadas a través de SNMP. Dichos umbrales deben ser justificados con base en las características de software y hardware y proporcionan alertas una vez que han sido sobrepasados.
2. La segunda parte es el algoritmo de Mínimos Cuadrados, el cual nos da una línea recta que es el resultado de una predicción con base a una colección de datos que nso pretende proyectar el crecimiento de los mismos, en este caso nuestra colección de datos se basa en el uso de la CPU.
3. Por último, la tercera parte correspondió al algoritmo de Holt Winters el cual se encargaba de identificar dentro de una gráfica no lineal, si los valores medidos salían de cierto rango de medición tanto superior como inferior y cuando esto sucedía, se notificaba por medio de un correo electrónico al administrador de la red.

En el capítulo mostrado a continuación se observa el desarrollo de la práctica indicando tanto el código utilizado para la implementación de la práctica como las pantallas que muestran paso a paso el proceso que se realizó para la aplicación de los algoritmos y los resultados obtenidos a partir de estos.

2.1. Algoritmo de Línea de Base

Línea base es un proceso para estudiar la red a intervalos regulares para asegurarse de que la red está trabajando dentro de los parámetros normales. Permite obtener información valiosa como:

- Salud de hardware y software.
- Uso de recursos de red.
- Umbrales de alarma.
- Problemas futuros en la red.

El proceso de línea base parte de un acuerdo de nivel de servicio (SLA) establecido entre el proveedor de servicio y sus clientes sobre el nivel de rendimiento esperado en los servicios de red. Es decir, en el SLA se establecen las métricas y valores configurados de manera realista, significativa y cuantificable para ambas partes.

Estas métricas son conformadas por estadísticas recogidas de los dispositivos de red para medir el nivel de rendimiento. Pueden incluir, uso de la CPU, la asignación de buffer y la asignación de memoria así como tráfico de red, capacidad de almacenamiento, etcétera [3].

Para realizar un análisis y ajuste del rendimiento primero se implementó un inventario necesario para conocer la cantidad de recursos disponibles en los hosts de la red. Eso se puede ver en la figura 2.1



IP del dispositivo	Nombre del dispositivo	Version de SO	Tiempo de actividad	Fecha y hora del host	Procesos en ejecucion
--------------------	------------------------	---------------	---------------------	-----------------------	-----------------------

Figura 2.1: Inventario.

En esta práctica se optó por recolectar las estadísticas vía SNMP de los siguientes recursos:

- Porcentaje de memoria RAM en uso con OID 1.3.6.1.4.1.2021.4.6.0.
- Porcentaje de espacio en disco duro en uso (HDD) 1.3.6.1.2.1.25.2.3.1.6.1.
- Porcentaje de uso de CPU por núcleo 1.3.6.1.2.1.25.3.3.1.2.

Así mismo, se estableció un SLA en el que se indican 3 umbrales para los tres recursos seleccionados. A continuación podemos observar el uso en el programa desarrollado por el equipo en la figura 2.2



Figura 2.2: Umbrales de host.

En la columna central de la ventana **Graficos** podemos observar tres botones: **Línea base - RAM**, **Línea base - HDD** y **Línea base - CPU**. Cada uno de ellos muestra la gráfica correspondiente.

Para el botón de **Línea base - RAM**, el programa despliega la gráfica situada debajo de la ventada de **Graficos** en el extremo superior izquierdo. Respectivamente, el botón de **Línea base - HDD**, el programa despliega la gráfica situada debajo de la ventada de **Graficos** en el extremo superior derecho. Por último, al presionar el botón de **Línea base - CPU**, el programa despliega una gráfica por cada uno de los núcleos de procesador con los que cuenta el agente. Estas gráficas son las que se encuentran en la parte inferior de la figura anterior.

Los umbrales establecidos fueron los siguientes:

- 50 % como umbral Ready indicador de que el dispositivo puede necesitar atención en el futuro.
- 75 % como umbral Set indicador previo como alarma de inicio de una planeación para realizar reparaciones, actualizaciones o configuraciones nuevas.
- 90 % como umbral Go que indica falla o acción inmediata.

Dichos umbrales fueron establecidos con base en el comportamiento normal del agente. Es decir para los tres rubros (RAM, HDD y CPU) se utilizó como base un histórico del uso de recursos del hosts. De esta manera se definió el 50 % como umbral Ready ya que sólo en algunas ocasiones se alcanza este límite. Así mismo, el número de veces que se alcanza el umbral Set es menor e indica que es posible que el umbral Go sea alcanzado en un momento próximo. En este caso, podemos observar que tanto en RAM como en HDD el umbral Ready fue alcanzado pero sin llegar al umbral Set. En el caso del uso del CPU podemos observar

que en el núcleo 1, ningún umbral fue alcanzado. Sin embargo, el núcleo 2 tuvo un pico fuera de lo normal que superó al umbral Set pero que no alcanzó al umbral GO.

Por último, se implementó un mecanismo de alarma vía correo electrónico que se activa cada que un host sobre pasa alguno de los umbrales. Este mecanismo es activado y ejecutado automáticamente por el programa. A continuación se muestra, los correos enviados:

Alerta - MiniObservium

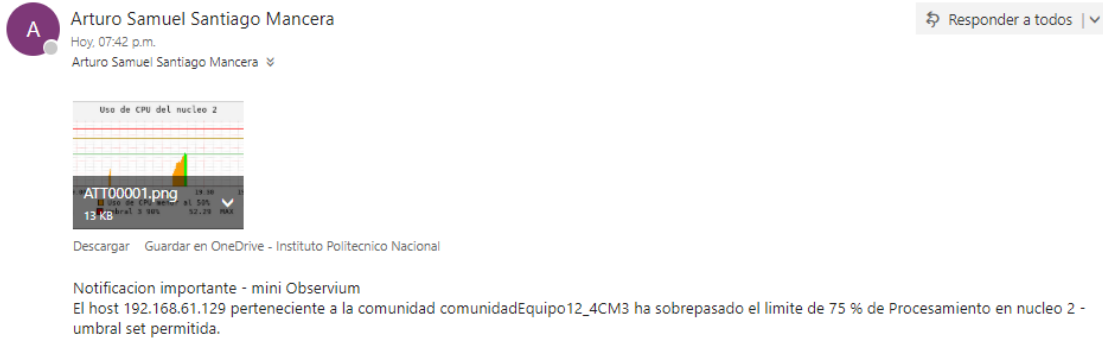


Figura 2.3: Correo 1.

Alerta - MiniObservium

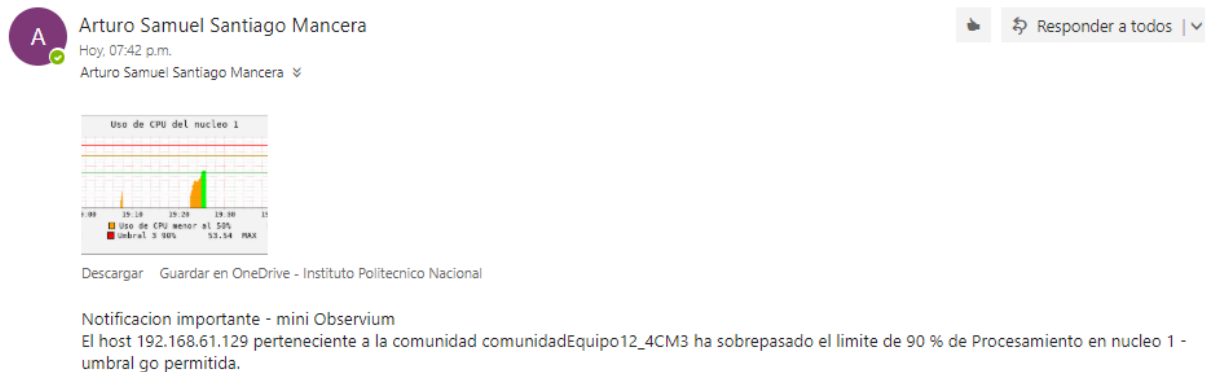


Figura 2.4: Correo 2.

Alerta - MiniObservium



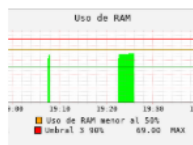
Arturo Samuel Santiago Mancera

Hoy, 07:42 p.m.

Arturo Samuel Santiago Mancera



Responder a todos



Descargar Guardar en OneDrive - Instituto Politécnico Nacional

Notificación importante - mini Observium

El host 192.168.61.129 perteneciente a la comunidad comunidadEquipo12_4CM3 ha sobrepasado el límite de 50 % de RAM - umbral ready permitida.

Figura 2.5: Correo 3.

2.1.1. Explicación de código

El código escrito para la implementación de línea base se basa casi en su totalidad en la recolección de datos SNMP con python vista en la **Práctica 1**. Esto abarca, la adquisición de datos y la actualización de las bases RRD ya que se utiliza la misma estructura RRD. No obstante, el código que si se puede destacar en esta práctica es el de la graficación. La figura 2.6 muestra la parte del código que permite graficar los umbrales.

```
317 def graficar_LB(cadena,rrd,image_name,id_grafica,limites,num_cores):
318     if id_grafica == 1:
319         while 1:
320             print cadena
321             ret = rrdtool.graphv(image_name,
322                                 "--title", "Uso de RAM",
323                                 "--start", str(tiempo_inicial),
324                                 "--end", str(tiempo_final),
325                                 "--vertical-label", "Uso %",
326                                 "--lower-limit", "0",
327                                 "--upper-limit", "100",
328                                 "DEF:ramused="+rrd+":ramused:AVERAGE",
329                                 "CDEF:ramA=ramused,1,*", # Ajuste de escala
330                                 "CDEF:ramU1=ramA,"+str(limites[0])+",GT,0,ramA,IF", #Si ramA es mayor que el primer umbral regresa 0, si no regresa ramA
331                                 "VDEF:ramMAX=ramA,MAXIMUM", # Toma el maximo valor entre todos los datos de ramA
332                                 "VDEF:ramMIN=ramA,MINIMUM", # Toma el minimo valor entre todos los datos de ramA
333                                 "VDEF:ramSTDEV=ramA,STDEV", # Toma la desviacion estandar de todos los datos de ramA
334                                 "VDEF:ramLAST=ramA,LAST", # Toma el ultimo valor existente en los datos de ramA
335                                 "AREA:ramA#00FF00:Uso de RAM", # Dibuja a ram con verde
336                                 "AREA:ramU1#FF0000:Uso de RAM menor al "+str(limites[0])+"%", # Dibuja el trafico menor al primer umbral
337                                 "HRULE:"+str(limites[0])+"#008000:Umbral 1 "+str(limites[0])+"%", # Dibuja un umbral
338                                 "HRULE:"+str(limites[1])+"#B18904:Umbral 2 "+str(limites[1])+"%", # Dibuja un umbral
339                                 "HRULE:"+str(limites[2])+"#FF0000:Umbral 3 "+str(limites[2])+"%", # Dibuja un umbral
340                                 "GPRINT:ramMAX:%6.2lf %SMAX", # Etiqueta
341                                 "GPRINT:ramMIN:%6.2lf %SMIN", # Etiqueta
342                                 "GPRINT:ramSTDEV:%6.2lf %SSTDEV", # Etiqueta
343                                 "GPRINT:ramLAST:%6.2lf %SLAST" ) # Etiqueta
344             time.sleep(1)
```

Figura 2.6: Código para generar gráficas.

Como se puede observar, se deben definir los 3 umbrales los cuales son recibidos como parámetros de la función **graficar_LB**. Así mismo, se definen las variables utilizadas, colores y etiquetas.

Otro segmento de código a destacar es el momento en el que se mandan las alertas vía correo electrónico. Esto se lleva a cabo en la adquisición de datos justo después de realizar la consulta SNMP con el OID correspondiente. Una vez obtenido este dato se compara con cada uno de los **limites** o umbrales establecidos. Esto se puede observar en la siguiente figura:


```

111 #-----
112 #Adquisicion de RAM
113 ram_used = int(consultaSNMP(comunidad,host,puerto,'1.3.6.1.4.1.2021.4.6.0')) #Memoria libre
114 porcentaje = (ram_used*100)/max_ram # Porcentaje de RAM libre
115 porcentaje = 100 - porcentaje; # Porcentaje de RAM usada
116 valor = "N:" + str(porcentaje)
117
118 #Verifica limites para enviar notificaciones
119 if porcentaje >= limites[2] and estados_limites[2] == 0:
120     estados_limites[2] = 1
121     notificar(host,comunidad,'% de RAM - umbral go',limites[2],',',str(1))
122 elif porcentaje >= limites[1] and estados_limites[1] == 0:
123     estados_limites[1] = 1
124     notificar(host,comunidad,'% de RAM - umbral set',limites[1],',',str(1))
125 elif porcentaje >= limites[0] and estados_limites[0] == 0:
126     estados_limites[0] = 1
127     notificar(host,comunidad,'% de RAM - umbral ready',limites[0],',',str(1))
128

```

Figura 2.7: Verificación de alertas .

Una vez que se ha sobre pasado uno de los umbrales se llama a la función **notificar**:

```

223 def notificar(host,comunidad,unidad,limite,id_nucleo,id_grafica):
224     #Implementa envio de correo electronico
225     cad = "Notificacion importante - Mini Observium:El host "+host+" perteneciente a la comunidad "+comunidad+" ha sobrepasado el limite de "+str(limite)+" "+unidad+" permitida."
226     #print cad + " " + host+" "+id_grafica+"-LB.png"+id_nucleo
227     enviaCorreo.LB(cad,host+" "+id_grafica+"-LB.png"+id_nucleo)

```

Figura 2.8: Llamada a notificar .

Esta función se encarga de formar la cadena correcta que se integrará al correo y a su vez, llama a la función **enviaCorreo.LB** la cual es la encargada de adjuntar la imagen de la gráfica correspondiente y enviar el correo:

```

19 def enviaCorreo.LB(mensaje,imagen):
20     print "Enviando correo"
21     msg = MIMEMultipart()
22     msg['From']='asantiagom1401@alumno.ipn.mx'
23     msg['To']="asantiagom1401@alumno.ipn.mx"
24
25     msg['Subject']="Alerta - MiniObservium"
26     msg.attach(MIMEText(mensaje))
27     #print png_file
28     fp = open(imagen, 'rb')
29     img = MIMEImage(fp.read())
30     fp.close()
31     msg.attach(img)
32
33     mailServer = smtplib.SMTP('smtp.office365.com',587)
34
35     mailServer.ehlo()
36     mailServer.starttls()
37     mailServer.ehlo()
38     mailServer.login('asantiagom1401@alumno.ipn.mx','<password>')
39
40     mailServer.sendmail('asantiagom1401@alumno.ipn.mx', "asantiagom1401@alumno.ipn.mx", msg.as_string())
41
42     mailServer.close()
43

```

Figura 2.9: Envío de correo.

2.2. Algoritmo de Mínimos Cuadrados

Para implementar el algoritmo de mínimos cuadrados y poder sacar la predicción de la pendiente se desarrollaron 3 módulos en el algoritmo las cuales son: aplicación, notificación y operación. En el módulo de **aplicación** básicamente es la estructura de la interfaz de la aplicación, el módulo de **notificación** manda el mensaje junto con la imagen de la gráfica resultante del algoritmo y por último el módulo de **operación** es en sí todo el desarrollo del algoritmo de mínimos cuadrados.

- **Aplicación** Empieza por la adquisición de información como se puede ver en la siguiente imagen [2.10](#)



Figura 2.10: Inicio de la interfaz de mínimos cuadrados.

La petición de datos implica: 1. La base de datos RRD con la que se va a trabajar el algoritmo de predicción. 2. El límite superior, en donde ese valor se intersecta con la recta definida con el algoritmo de predicción. 3. La fecha de inicio, a partir de este valor, comenzará a evaluar nuestro algoritmo, no será necesario introducir la fecha de la última actualización porque nuestro algoritmo lo puede resolver, dicho método se explicará posteriormente. 4. Data Source, es la variable en donde la base de datos comenzará a recuperar información, será importante para generar la gráfica y asociar el algoritmo de predicción con dicha variable.

```
def widgets(self):
    self.style = ttk.Style()
    self.style.configure("LBL.TLabel", foreground="White", background=Background_, font='verdana 12')
    self.style.configure("LBL2.TLabel", foreground="#ed6f09", background='#2b2933', font='verdana 12')
    self.style.map("BTN.TButton",
        foreground=[('pressed', '#6f16ce'), ('active', 'white')],
        background=[('pressed', 'disabled', '#726e77'), ('active', '#38c0ed')]
    )
    self.style.configure("BTN.TButton", font='verdana 12', foreground='white', background='#38c0ed')
    self.style.configure("SEP.TSeparator", background='black')
    #self.style.configure("BTN.TButton", foreground="white", background="#38c0ed", font='verdana 12')
    self.separador = ttk.Separator(orient=HORIZONTAL, style="SEP.TSeparator")
    self.separador2 = ttk.Separator(orient=HORIZONTAL, style="SEP.TSeparator")
    self.separador3 = ttk.Separator(orient=HORIZONTAL, style="SEP.TSeparator")

    self.text_1 = Text(self.raiz, width=60, height=1)
    self.text_1.config(state=DISABLED)
    self.entry_2 = ttk.Entry(self.raiz, textvariable=self.txt_limite)
    self.entry_3 = ttk.Entry(self.raiz, textvariable=self.txt_fecha_inicio)
    self.entry_4 = ttk.Entry(self.raiz, textvariable=self.txt_ds)

    self.btn_archivo = ttk.Button(self.raiz, text='...', style="BTN.TButton", command=lambda: self.seleccionar_archivo(), width=3)
    self.lbl_1 = ttk.Label(self.raiz, text='Selecciona archivo rrd', style="LBL.TLabel", width=20)
    self.lbl_2 = ttk.Label(self.raiz, text='Limite (0 - 100)', style="LBL.TLabel", width=20)
    self.lbl_3 = ttk.Label(self.raiz, text='Fecha Inicio (Unix time)', style="LBL.TLabel", width=20)
    self.lbl_4 = ttk.Label(self.raiz, text='Data Source', style="LBL.TLabel", width=20)
    self.btn_ayuda = ttk.Button(self.raiz, text='?', style="BTN.TButton", command=lambda: self.ayuda(), width=3)
    self.btn_evaluar = ttk.Button(self.raiz, text='Evaluar', style="BTN.TButton", command=lambda: self.Evaluar())
```

Figura 2.11: Ingreso de valores.

Se hacen las validaciones pertinentes como se muestra en la imagen 2.12 para que no haya ningún campo sin llenar, tanto la ruta del archivo como los valores deben ser ingresados o por su defecto son los valores, que se tienen por default. En la línea 85 se evalúa la hora estimada en la que se pronostica que el fallo podrá ocurrir, en determinada forma es la función más importante ya que en ella se evalúa el método de mínimos cuadrados y se regresa el valor en tiempo UNIX, como conclusión tenemos ese valor en la variable “estimado”.

En la línea 86 se manda llamar la función que nos generará la gráfica con los valores, enviando así mismo la hora estimada para que pueda trazar una línea vertical representando con mayor precisión la hora de la predicción del fallo.

Si la respuesta es Error, quiere decir que ha ocurrido un error al momento de graficar

```

57 def Evaluar(self):
58     ruta = self.txt_ruta_archivo.get()
59     inicio = 0
60     limite = 0
61     ds = self.txt_ds.get()
62     if ruta == '':
63         self.txt_ruta_imagen.set('')
64         print('Error')
65         return
66     self.txt_ruta_imagen.set(self.txt_ruta_archivo.get().split('.')[0] + '.png')
67     try:
68         inicio = int(self.txt_fecha_inicio.get())
69         limite = int(self.txt_limite.get())
70     except:
71         print('Error')
72         return
73
74     if limite < 0 or limite > 100:
75         print('Error')
76         return
77     if ds == '':
78         print('Error')
79         return
80
81     self.graficar(ruta, limite, inicio, ds)
82
83 def graficar(self, ruta, limite, inicio, ds):
84     self.raiz.withdraw()
85     estimado = notify.check_CPU2(ruta, ds, limite, inicio) #int
86     respuesta = operaciones.graficarLectura(ruta, str(inicio), str(limite), ds, estimado)
87     if respuesta == 'Error':
88         print('Se ha generado un Error, línea 87')
89     return
90

```

Figura 2.12: Validación de campos de datos.

Posteriormente sigue el código como se ve en la imagen 2.13 que presenta la información, que consta de: Una variable llamada **strinicio** declarada en la línea 96 la cual nos va a imprimir en Una etiqueta la fecha en la que se comenzó a evaluar los valores de la base de datos.

Una variable llamada **strfin** la cual determina en que momento se tomó el último valor en la base de datos.

Una variable llamada **strpred** el cual representa la hora exacta en la que el fallo está pronosticado, este valor lo tomamos de la variable “estimado”.

Finalmente una imagen la cual fué generada en la función: **graficarLectura** de la línea 86.

```

82
83 def graficar(self, ruta, limite, inicio, ds):
84     self.raiz.withdraw()
85     estimado = notify.check_CPU2(ruta, ds, limite, inicio) #int
86     respuesta = operaciones.graficarLectura(ruta, str(inicio), str(limite), ds, estimado)
87     if respuesta == 'Error':
88         print('Se ha generado un Error, linea 87')
89         return
90
91     ventana = Toplevel()
92     ventana.geometry('800x300+150+100')
93     ventana.title('Prediccion Minimos Cuadrados')
94     ventana.configure(background=Background)
95     try:
96         str_inicio = str(time.ctime(inicio))
97         str_fin = str(time.ctime(rrdtool.last(ruta)))
98         str_pred = str(time.ctime(estimado))
99
100         self.pic = PhotoImage(file=self.txt_ruta_imagen.get())
101         imagen = ttk.Label(ventana, image=self.pic)
102
103         lbl_1 = ttk.Label(ventana, text='Inicio de Captura:', style='LBL.TLabel')
104         lbl_2 = ttk.Label(ventana, text=str_inicio, style='LBL2.TLabel')
105
106         lbl_3 = ttk.Label(ventana, text='Ultima Captura:', style='LBL.TLabel')
107         lbl_4 = ttk.Label(ventana, text=str_fin, style='LBL2.TLabel')
108
109         lbl_5 = ttk.Label(ventana, text='Prediccion Estimada:', style='LBL.TLabel')
110         lbl_6 = ttk.Label(ventana, text=str_pred, style='LBL2.TLabel')
111
112         btn_cerrar = ttk.Button(ventana, text='Volver', command=lambda:self.cerrar(ventana), style='BTN.TButton')
113
114         imagen.grid(column=0, row=0, rowspan=7, pady=10, padx=10)
115         lbl_1.grid(column=1, row=0)
116         lbl_2.grid(column=1, row=1)
117         lbl_3.grid(column=1, row=2)
118         lbl_4.grid(column=1, row=3)

```

Figura 2.13: Función graficar.

• Notificación

La función CPU es la más importante ya que al final del proceso regresará la fecha en tiempo UNIX de la predicción que queremos realizar, dicho de otro modo, al registrar los valores de carga de CPU queremos saber, por ejemplo, en qué momento llegará esa carga al 60 por ciento, entonces para solucionar este problema se implementa el algoritmo de mínimos cuadrados, el cual es utilizado de la siguiente manera:

Declaramos la variable info para saber cada cuanto tiempo se realiza un step en la base de datos, ese valor se almacena en la variable rrdstep. La variable estimado tendrá el valor final en el que la predicción coincide con el porcentaje de carga de CPU al que queremos evaluar. Inicialmente comienza desde la fecha de inicio de captura de datos más el step que sería equivalente a la primer captura.

Posteriormente declaramos un nombre para un archivo temporal ya que no queremos que por el momento nos genere una gráfica, más bien, queremos aprovechar las bondades de la función graph para poder realizar el algoritmo de mínimos cuadrados.

Posteriormente viene un while infinito, se detendrá hasta que la predicción supere el 100 por ciento o encuentre el valor que estamos buscando.

Realizamos el método graph en el que como inicio se declara el inicio de la captura de datos de la base RRD, como fecha final se coloca el valor estimado, tiene la finalidad de que en ese rango que será variable, evalúe mínimos cuadrados y el valor final compararlo con el valor que esperamos obtener.

El algoritmo se realiza de la siguiente manera, tomando en cuenta que la predicción de mínimos cuadrados se basa en la recta $y = mx + b$: definimos la variable carga la cual tendrá la colección de datos de la carga de CPU desde el inicio hasta el valor estimado que en primera instancia solo es de un step. La variable a utiliza la función **LSLSLOPE** que representa la pendiente de la recta (m) utilizando la colección de valores. La variable b utiliza la función **LSLINT** que representa la ordenada al origen (b). Avg2 realiza el algoritmo de obtención del valor Y, lo que hace es extraer los valores de carga con

un POP, tiene la variable y los multiplica, teniendo así: $mx + b$ de la ecuación inicial. Posteriormente le suma b , teniendo así $mx + b$, como resultado tenemos el valor en Y de ese rango de valores.

Realizamos la función PRINT para poder recuperar el último valor del algoritmo. Entonces en conclusión cada iteración se evalúa a un tiempo estimado la última evaluación del método de mínimos cuadrados, por lo que de obtener el valor esperado tendremos también la fecha esperada y será el valor de retorno.

Dentro del bloque try realizamos un cast del último valor capturado en el método de mínimos cuadrados y evaluamos que si es mayor o igual al valor que esperamos nos regrese la fecha estimada, si no es así, si el valor es mayor a 100 encontramos un error y regresar la última fecha de captura de información.

Si ninguno de estos casos se cumple, entonces aumentamos nuestra variable estimado en un step más. Por tanto recorreremos a cada paso el algoritmo hasta encontrar nuestro valor esperado. Se pueden ver en las imágenes 2.14 y 2.15

```
def check_CPU2(name, ds, upper, inicio):
    print('checking....')

    info = rrdtool.info(name)
    rrdstep = int(info['step'])
    estimado = int(inicio) + rrdstep
    graphtmpfile = tempfile.NamedTemporaryFile()
    while 1:
        #print('Hora estimada = ' + str(ctime(estimado)) + ' - ' + str(estimado))
        values = rrdtool.graph(graphtmpfile.name + 'F', #name.split('.')[0] + '.png',
                               "--start", str(inicio),
                               "--end", str(estimado),
                               "DEF:carga=" + name + ':' + ds + ':LAST', # trend.rrd:CPUload:AVERAGE
                               "VDEF:a=carga,LSSL Slope",
                               "VDEF:b=carga,LSLINT",
                               "CDEF:avg2=carga,POP,a,COUNT,*,b,+",
                               "PRINT:avg2:LAST:%1.0lf")

        #print(values)
        try:
            fail = int(values[2][0])
            if int(fail) >= int(upper):
                print('Encontramos la falla en:' + str(estimado))
                return estimado
            else:
                if int(fail) > int(100):
                    print('ERROR :(' + str(estimado - 150759900))
                    return int(rrdtool.last(name))
        except:
            print('Sin valores')
        estimado = estimado + int(rrdstep)
```

Figura 2.14: Función checkCPU.

```
        "CDEF:avg2=carga,POP,a,COUNT,*,b,+",
        "PRINT:avg2:LAST:%1.0lf")

    #print(values)
    try:
        fail = int(values[2][0])
        if int(fail) >= int(upper):
            print('Encontramos la falla en:' + str(estimado))
            return estimado
        else:
            if int(fail) > int(100):
                print('ERROR :(' + str(estimado - 150759900))
                return int(rrdtool.last(name))
    except:
        print('Sin valores')
    estimado = estimado + int(rrdstep)
```

Figura 2.15: Validación de la función checkCPU.

• Operación

La función **GraficarLectura** recibe 5 parámetros los cuales son: 1) el nombre de la base de datos 2) la hora de inicio de captura de datos 3) el límite en donde se realizará la estimación en porcentaje, por ejemplo, 60 por ciento de la carga de CPU 4) el data source de la base de datos 5) la fecha en tiempo UNIX que el algoritmo de predicción regresó

Se ocupa la función **graph** de rrdtool la cual indica: el nombre del archivo que va a generar, en este caso es una imagen png. la fecha de inicio y final en la cual graficará, que la hora final se especificó 10 minutos después del fallo para poder apreciar bien la línea que marca la intersección con el fallo. Definimos los límites y una variable llamada carga que obtiene una lista con los últimos valores capturados en la base de datos a través del tiempo. Posteriormente, se declara una línea horizontal con el valor establecido con el cual queremos realizar la predicción. Realizamos el algoritmo de mínimos Cuadrados con el fin de poder apreciar la línea recta que genera. Finalmente se utiliza **VRULE** para trazar una línea vertical en una fecha específica, esta línea representa el fallo que se calculó previamente, por lo que el resultado final imprime una línea vertical a la hora del posible próximo fallo con el fin de apreciarlo gráficamente.

Si todo el proceso salió bien regresa '**Exito**', de lo contrario regresa '**Error**'. El código se puede ver en la imagen 2.16

```
def graficarLectura(nombre, hora, upper, ds, fallo2):

    #fallo2 = int(notify.check_CPU2(nombre, ds, upper, hora))
    str_fallo = ctime(fallo2).split(':')

    hora_final = str(fallo2 + 600) # str(int(hora) + int(fallo) ) #rrdtool.last(nombre) + 15000

    try:
        ret = rrdtool.graph(nombre.split('.')[0] + '.png',
            "--start", hora,
            "--end", hora_final,
            "--vertical-label=uso CPU",
            "--title=Uso de CPU",
            "--color", "ARROW#009900",
            "--vertical-label", "Uso de CPU (%)",
            "--lower-limit", '0',
            "--upper-limit", '100',
            "DEF:carga=" + nombre + ':' + ds + ':LAST', # trend.rrd:CPUload:AVERAGE
            "AREA:carga#00FF00:CPU usage",

            "LINE1:" + upper + "#FF0000: Upper Limit", # "LINE1:50#FF0000: Limite"
            "VDEF:a=carga,LSLSLOPE",
            "VDEF:b=carga,LSLINT",
            "CDEF:avg2=carga,POP,a,COUNT,*,b,+",
            # "GPRINT:avg2:AVERAGE:%7.0lf",
            # "PRINT:avg2:AVERAGE:%1.0lf",
            "LINE2:avg2#FFB000:MMC",
            "VRULE:" + str(fallo2) + "#f72ce9:Fallo",
            "COMMENT: Prediction Detected at " + str_fallo[0] + '.' + str_fallo[1] + '.' +
            str_fallo[2])

        return 'Exito'
    except:
        return 'Error'
```

Figura 2.16: Función graficar.

Finalmente el ejercicio a desarrollar en la práctica 2 se trato de sacar la recta de predicción de mínimos cuadrado con base a los datos obtenidos en el archivo **source3.rrd** con un límite de umbral de un **90** y la fecha de inicio **1539659123** la cual esta en formato Unix y por último el archivo data source **CPUload** como se muestra en la imagen 2.26

Prediccion Lineal

Selecciona archivo rrd
/home/arturo/Escritorio/source3.rrd

Limite (0 - 100)
90

Fecha Inicio (Unix time)
1539659123

Data Source
CPUload

Evaluar

Figura 2.17: Valores para iniciar la grafica de Mínimos cuadrados.

Como resultado de los valores ingresados se obtuvo la siguiente gráfica que nos calcula el momento exacto en donde la predicción de mínimos cuadrados rebasa el umbral dándonos el día y hora de dicho cruce.

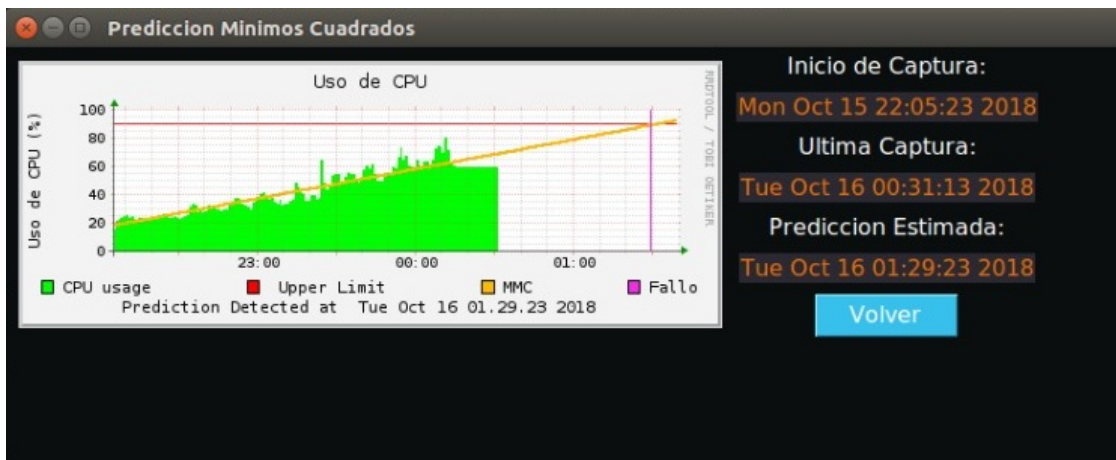


Figura 2.18: Gráfica resultante del algoritmo de mínimos cuadrados.

Nos da como resultado el día **jueves 16 de octubre del 2018 a la 1:29:23 hrs.**

2.3. Algoritmo de Holt Winters

El método Holt-Winters es un método de pronóstico de triple exponente suavizante y tiene la ventaja de ser fácil de adaptarse a medida que nueva información real está disponible. El método Holt-Winters es una extensión del método Holt que considera solo dos exponentes suavizantes. Holt-Winters considera nivel, tendencia y estacional de una determinada serie de tiempos.

El método de Holt-Winters es básicamente un procedimiento de suavizamiento exponencial. Este tipo de procedimientos facilitan los cálculos y reducen los requerimientos de almacenamiento en las bases de datos, lo cual cobra importancia cuando se están prediciendo muchas series de tiempo [1].

Existen tres fases de trabajo, con tres conjuntos de datos diferentes.

- Un primer grupo de datos es para inicializar el modelo, esto es determinar los indicadores de nivel, tendencia y estacionalidad.
- Un segundo conjunto de datos es necesario para probar o calibrar los índices de suavización Alfa, Beta y Gamma.
- Un tercer grupo de datos para pronosticar y evaluar el funcionamiento del modelo propuesto.

Ejecutar todas las fases en un solo grupo de datos puede conducir a tratar de encajar en exceso el modelo a los datos disponibles [2].

Una vez que ya se ha explicado a grandes rasgos el funcionamiento del algoritmo de Holt Winters, se explicará paso a paso el procedimiento de la aplicación de dicho algoritmo.

Primero se añadió el agente desde el cual se obtuvieron los datos en tiempo real tal y como se muestra en la figura 2.19.

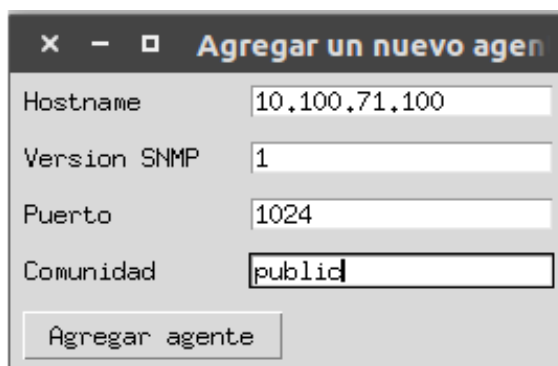


Figura 2.19: Nuevo agente añadido.

Y de los agentes añadidos, se mostró la información en la pantalla principal como se observa en la figura 2.20. En esta imagen no se observa que el agente se encuentre activo debido a que su ejecución se realizó directamente en el laboratorio, sin embargo, en la figura 2.21 se muestran los datos del host 10.100.71.100 cuando este se encontraba activo.

Bienvenido a casi Observium :)			
Agregar agente		Dispositivos monitoreados	
Eliminar agente		Numero de agentes: 2	
Nombre del agente		Status	No. de interfaces
Linux ranch-Lenovo-Y50-70 4.15.0-36-generic #39~16.04.1-Ubuntu SMP Tue Sep 25 08:59:23 UTC 2018 x86_64		Activa	3
10.100.71.100		Inactiva	
		Nombre interfaz	Status interfaz
		lo	Activo
		Realtek Semiconductor Co., Ltd. RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller	Inactivo
		Realtek Semiconductor Co., Ltd. RTL8723BE PCIe Wireless Network Adapter	Activo
		Informacion no disponible	

Figura 2.20: Pantalla principal de agentes.

Linux Ev2 4.8.6.5-smp #2 SMP Vie Oct 26 14:58:11 CDT 2016 i686		
Activa	2	lo eth0

Figura 2.21: Información del agente activo.

Una vez que el agente fue añadido, en los botones que se encuentran a lado derecho se pulsó sobre el botón con la leyenda **Gráficos**, mismo que desplegaba una ventana con las opciones siguientes (figura 2.22):

Servicio		Algoritmos	
Trafico de la interfaz	Linea de base - CPU	Holt Winters OutNUcastPkts	
Conexiones TCP establecidas	Linea de base - RAM	Holt Winters conexiones TCP	
Segmentos TCP	Linea de base - HDD	Holt Winters segmentos TCP	
Estadísticas ICMP		Holt Winters estadísticas ICMP	
Respuestas SNMP		Holt Winters respuestas SNMP	

Figura 2.22: Ventana de gráficos disponibles.

Al ser el equipo 10, nos fue asignado el monitoreo del OID de los **OutNUcastPkts**, por tal motivo se presionó sobre el primer botón y al realizar dicha acción se comenzó el monitoreo de los paquetes mostrando una gráfica similar a la de la figura 2.23.

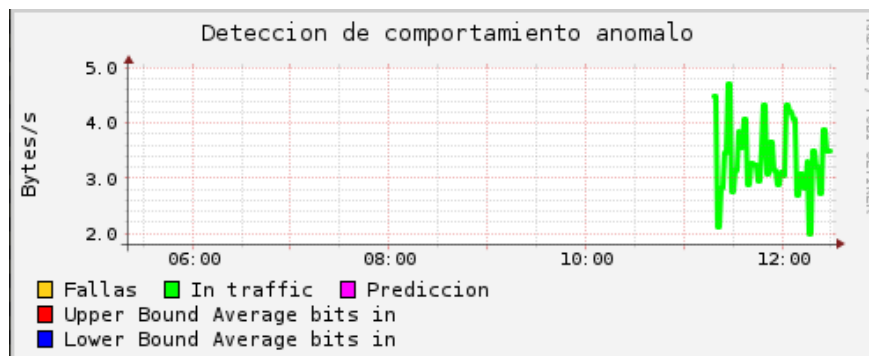


Figura 2.23: Gráfica de OutNUcastPkts.

Misma que posteriormente y conforme fueron surgiendo los distintos fallos se mostró como las gráficas de a continuación:

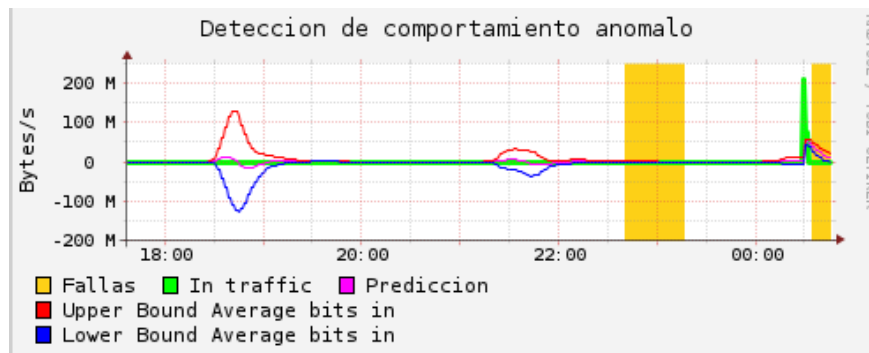


Figura 2.24: Gráfica de OutNUCastPkts.

Sin embargo, también se realizaron mediciones con otros paquetes tal y como la gráfica de la figura 2.25 en la cual se plasman los datos obtenidos con el OID de **InOctets**.

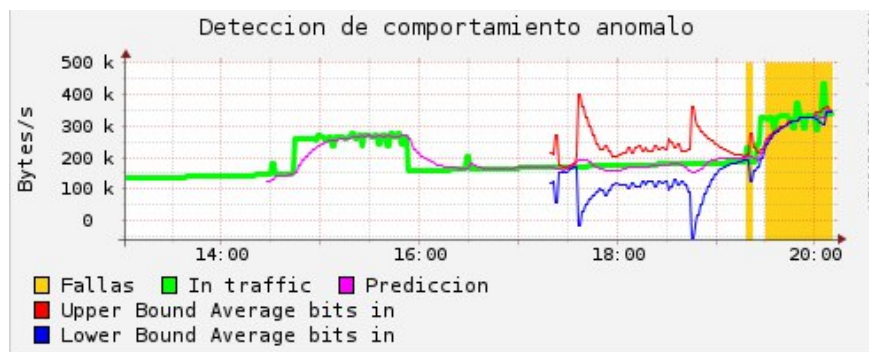


Figura 2.25: Gráfica de InOctets.

Y como se observa en las gráficas mostradas anteriormente, una línea amarilla vertical marca las secciones en las cuales se ha sobrepasado el límite inferior o superior. Cuando dicho error sucede, se envían dos correos electrónicos al administrador indicando tanto el inicio del error como el final del mismo y adjuntando en dicho correo las gráficas como se puede observar en las figuras a continuación:

Respecto a los datos de entrada de OutNUCastPkts entre los correos recibidos se observan los de las figuras 2.26 y 2.27.

New aberrations detected ➤ Recibidos x



ruben.murga.d@gmail.com



March Castro <march.castrof@gmail.com>
para tanibet.escom ▼

----- Forwarded message -----

From: <ruben.murga.d@gmail.com>

Date: vie., 26 oct. 2018 a las 11:47

Subject: New aberrations detected

To: <march.castrof@gmail.com>

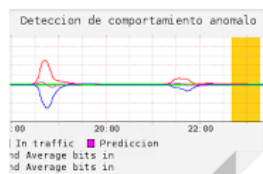


Figura 2.26: Correo de inicio del error.

Abberations gone ➤ Recibidos x



ruben.murga.d@gmail.com



March Castro <march.castrof@gmail.com>
para tanibet.escom ▼

----- Forwarded message -----

From: <ruben.murga.d@gmail.com>

Date: vie., 26 oct. 2018 a las 11:48

Subject: Abberations gone

To: <march.castrof@gmail.com>

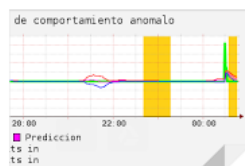


Figura 2.27: Correo de fin del error.

Por otro lado, respecto a los datos de entrada de InOctets entre los correos recibidos se observan los de las figuras 2.28 y 2.29.

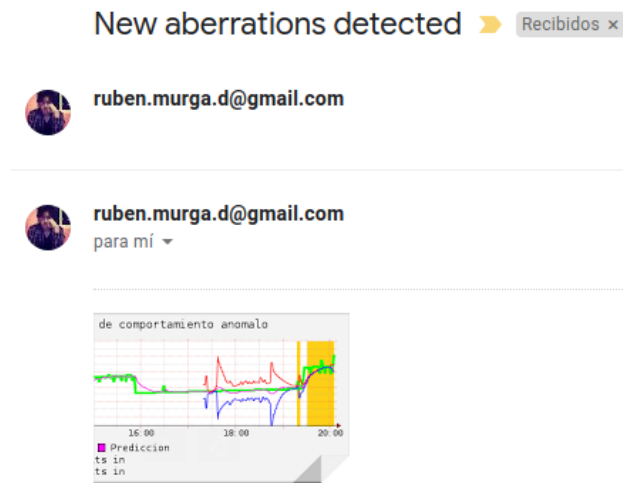


Figura 2.28: Correo de inicio del error.

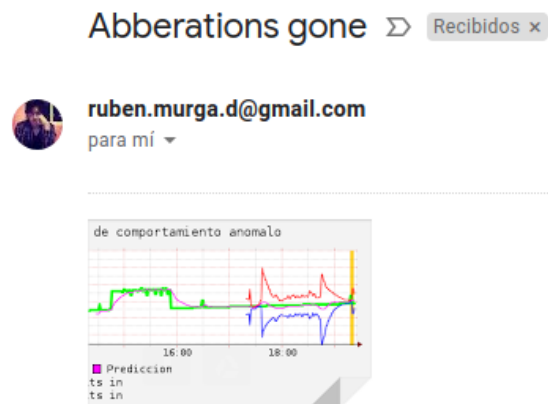


Figura 2.29: Correo de fin del error.

2.3.1. Explicación de código

En esta sección se explicarán las partes del código más importante. En el método **actualizarHW** mostrado en la figura 2.30, lo primero que se realiza es al momento de obtener por parámetros los datos del host al cual se desea graficar, se verifica si existe ya un archivo .rrd en el cual ya se almacenen los datos, en caso contrario se manda a llamar al método **crearHW**.

```
def actualizarHW(cadena, comunidad, host, puerto, rrd):
    print 'Entro actualizarHW ', cadena, comunidad, host, puerto, rrd

    total_input_traffic = 0
    total_output_traffic = 0
    rrdpath="./RRD_HW/"
    pngpath="./IMG_HW/"
    fname=rrd+".rrd"
    pngfname=rrd+".png"
    #Verifica que exista una rrd asociada al host, en caso de no e
    archivo_rrd = Path(rrdpath+fname)
    if archivo_rrd.is_file() == False:
        crearRRD.crearHW(rrdpath+fname)
        print "rrd HW Creada... en", rrdpath, fname
    else:
        print "Abriendo rrd HW..."
```

Figura 2.30: Método actualizarHW.

El método **crearHW**, mostrado en la figura 2.31 se encarga de crear el archivo .rrd en el cual se almacenarán los datos indicados que en este caso se refieren a los datos de entrada de OutNUCastPkts.

```
def crearHW(nombre):
    print 'NOMBREEE '+nombre

    ret = rrdtool.create(nombre,
        "--start", 'N',
        "--step", '60',
        "DS:outucastpkts:COUNTER:600:U:U",
        "RRA:AVERAGE:0.5:1:1209",
        "RRA:HWPREDICT:600:0.9:0.0035:172:3",
        "RRA:SEASONAL:172:0.9:2",
        "RRA:DEVSEASONAL:172:0.9:2",
        "RRA:DEVPREDICT:600:4",
        "RRA:FAILURES:172:7:9:4")
    if ret:
        print rrdtool.error()
```

Figura 2.31: Método crearHW.

Una vez que se ha creado la base rrd, volvemos al código completo del método **actualizarHW** mostrado en la figura 2.32, en el cual se obtienen los datos de entrada que se almacenan en la variable *total_input_traffic*, utilizando el protocolo SNMP al cual se indica el OID, hostname, puerto y nombre de la comunidad y con dichos datos obtenidos, se actualiza el archivo .rrd y con el comando .dump, se envían esos datos a un archivo .xml que es legible para el usuario.

```
def actualizarHW(cadena,comunidad,host,puerto,rrd):
    print 'Entro actualizarHW ',cadena,comunidad,host,puerto,rrd

    total_input_traffic = 0
    total_output_traffic = 0
    rrdpath="./RRD_HW/"
    pngpath="./IMG_HW/"
    fname=rrd+".rrd"
    pngfname=rrd+".png"
    #Verifica que exista una rrd asociada al host, en caso de no existir crea una rrd nueva
    archivo_rrd = Path(rrdpath+fname)
    if archivo_rrd.is_file() == False:
        crearRRD.crearHW(rrdpath+fname)
        print "rrd HW Creada... en",rrdpath,fname
    else:
        print "Abriendo rrd HW..."

    endDate = rrdtool.last(rrdpath+fname) #ultimo valor del XML
    begDate = endDate - 3600
    #Inicia proceso de adquisicion de datos HW
    while 1:
        total_input_traffic = int(consultaSNMP('public','10.100.71.100',1024,'1.3.6.1.2.1.2.2.1.18.1'))
        total_output_traffic = int(consultaSNMP(comunidad,host,puerto,'1.3.6.1.2.1.2.2.1.16.3'))
        valor = str(rrdtool.last(rrdpath+fname)+30)+":"+str(total_input_traffic)
        print 'Valor: ',valor
        rrdtool.update(rrdpath+fname,valor)
        rrdtool.dump(rrdpath+fname,rrd+'.xml')
        #print "actualizar: ",rrdpath,fname, rrd
        rrdtool.tune(rrdpath+fname,'--alpha','0.1')
        #time.sleep(1)
    if ret:
        print rrdtool.error()
        time.sleep(300)
```

Figura 2.32: Método actualizarHW.

Posteriormente, la figura 2.33 muestra el código utilizado para realizar la graficación de los valores que se están leyendo desde el archivo .rrd. En este método se indican que valores se desean graficar, los colores y lo que significará cada una de las etiquetas mostradas en la parte inferior de la gráfica.

```
def graficar_Hw(cadena, rrd, image_name, id_grafica):
    print "GRAFICAR ", rrd, image_name
    rrdpath="./RRD_HW/"
    pngpath="./IMG_HW/"
    title="Deteccion de comportamiento anomalo"
    var = 0
    if id_grafica == 1:
        while 1:
            ret = rrdtool.graph(pngpath+image_name,
                                '--start', str(rrdtool.last(rrdpath+rrd)-25800),
                                #'--start', str(rrdtool.last(fname)-51600),
                                '--end', str(rrdtool.last(rrdpath+rrd)),
                                '--title=' + title,
                                '--vertical-label=OutUCastPkts/s',
                                '--slope-mode',
                                "DEF:obs=" + rrdpath+rrd + ":outucastpkts:AVERAGE",
                                "DEF:pred=" + rrdpath+rrd + ":outucastpkts:HWPREDICT",
                                "DEF:dev=" + rrdpath+rrd + ":outucastpkts:DEVPREDICT",
                                "DEF:fail=" + rrdpath+rrd + ":outucastpkts:FAILURES",

                                "CDEF:scaledobs=obs,8,*",
                                "CDEF:upper=pred,dev,2,*,+",
                                "CDEF:lower=pred,dev,2,*,-",
                                "CDEF:scaledupper=upper,8,*",
                                "CDEF:scaledlower=lower,8,*",
                                "CDEF:scaledpred=pred,8,*",

                                "TICK:fail#FDD017:1.0: Fallas",
                                "LINE3:scaledobs#00FF00:In traffic",
                                "LINE1:scaledpred#FF00FF:Prediccion\\n",
                                #"LINE1:outoctets#0000FF:Out traffic",
                                "LINE1:scaledupper#ff0000:Upper Bound Average bits in\\n",
                                "LINE1:scaledlower#0000FF:Lower Bound Average bits in")

            time.sleep(1)
            returned_value = check_aberration(rrdpath, rrd)
            print returned_value

            if var == returned_value:
                pass
            else:
                if var == 0 and returned_value == 1 or var == 2 and returned_value == 1:
                    send_alert_attached('New aberrations detected', pngpath+image_name)
                    var = returned_value
                elif var == 1 and returned_value == 2:
                    send_alert_attached('Abberations gone', pngpath+image_name)
                    var = returned_value
```

Figura 2.33: Método graficar.HW.

Por último, las figuras 2.34 y 2.35 muestran los métodos encargados de revisar los datos en búsqueda de un cambio de valores en los fallos y en caso de encontrarlo se realiza el envío del correo electrónico. El método `check_aberration` se encarga de verificar los datos que se generan en los fallos mismos que solo toman un valor de 0 o 1, en caso de que el valor haya cambiado de 0 a 1, significa que un fallo ha comenzado, si el valor cambia de 1 a 0, significará que el fallo ha finalizado.

```
def check_aberration(rrdpath, fname):
    """ This will check for begin and end of aberration
        in file. Will return:
        0 if aberration not found.
        1 if aberration begins
        2 if aberration ends
    """
    ab_status = 0
    rrdfilename = rrdpath + fname
    print rrdfilename
    info = rrdtool.info(rrdfilename)

    rrdstep = int(info['step'])
    print 'STEP', rrdstep
    lastupdate = info['last_update']
    print 'LASTUPD ', int(lastupdate)
    previosupdate = str(lastupdate - rrdstep - 1)
    graphtmpfile = tempfile.NamedTemporaryFile()

    try:
        values = rrdtool.graph(graphtmpfile.name+'F',
                               '--start', str(previosupdate),
                               '--end', str(lastupdate),
                               'DEF:f0=' + rrdfilename + ':outucastpkts:FAILURES',
                               'PRINT:f0:LAST:%1.0lf')

        print values

        if str(values[2][0]) == '-nan':
            print 'ERROR'
        else:
            flast = int(values[2][0])
            if (flast == 1):
                ab_status = 1
            else:
                ab_status = 2
        return ab_status
    except:
        pass
```

Figura 2.34: Método `check_aberration`.

Por otro lado, el método se encarga de enviar el correo correspondiente con el titular necesario para indicar si el fallo comenzó o finalizó y anexando la imagen de la gráfica en el punto en el cual comenzó o terminó dicho fallo.

```
def send_alert_attached(subject, file):  
    """ Will send e-mail, attaching png  
    files in the flist.  
    """  
    msg = MIMEMultipart()  
    msg['Subject'] = subject  
    msg['From'] = mailsender  
    msg['To'] = mailreceip  
    #for file in flist:  
    png_file = file  
    print png_file  
    fp = open(png_file, 'rb')  
    img = MIMEImage(fp.read())  
    fp.close()  
    msg.attach(img)  
  
    mserver = smtplib.SMTP('smtp.gmail.com', 587)  
    mserver.ehlo()  
    mserver.starttls()  
    mserver.ehlo()  
    mserver.login(mailsender, gmail_password)  
    mserver.sendmail(mailsender, mailreceip, msg.as_string())  
    mserver.close()
```

Figura 2.35: Método send_alert_attached.

CAPÍTULO 3

Cuestionario

1. ¿Cuál es el OID para conocer el uso del procesador?
Porcentaje de uso de CPU por núcleo 1.3.6.1.2.1.25.3.3.1.2.
2. ¿Cuál es el OID para obtener el uso de la memoria RAM?
Porcentaje de memoria RAM en uso con OID 1.3.6.1.4.1.2021.4.6.0.
3. ¿Cuál es el OID para sondear el uso del almacenamiento del dispositivo?
Porcentaje de espacio en disco duro en uso (HDD) 1.3.6.1.2.1.25.2.3.1.6.1.

Conclusiones

- **Castro Flores Marcela**

La realización de esta segunda práctica me pareció muy interesante porque fue una integración de lo que ya se había implementado en el primer parcial más las nuevas funcionalidades tales como notificar al administrador cuando un fallo ha ocurrido. Pienso que es importante el conocer este tipo de funcionalidades pues nos permite optimizar los servicios de las redes y también prevenir la situación en la cual un fallo pueda ocasionar un retraso en un proyecto por ejemplo pues esto podría ocasionar un retraso y un incremento en costos.

Así mismo, creo que es interesante el aprender nuevos algoritmos como los aplicados en este parcial ya que nos muestra una perspectiva de como aunque parezca que las matemáticas, la probabilidad y las redes informáticas pertenecen a áreas diferentes, en conjunto permiten la implementación de alternativas que evitan o minimizan el impacto de un fallo.

- **Sánchez Cruz Rosa María**

En el desarrollo de la práctica aprendimos una forma con la que pueden administrarse los sistemas y sus servicios, es importante tanto a nivel personal como a nivel empresa. A nivel personal podemos implementar un sistema que administre el calor que genera el procesador y poder realizar una acción para contrarrestar el problema. Por otro lado, en el ámbito empresarial se pueden administrar los servicios como lo es el número de conexiones, uso de espacio en disco, asociado claro a la especificación de lo que se va a brindar, por ejemplo un servidor web podrá contemplar cierta cantidad de usuarios los cuales no deberá sobrepasar, para esto se utilizan los algoritmos de predicción y las estrategias para evitar estos estados. Finalmente consideramos importante el desarrollo de la práctica ya que implementa una forma para controlar aspectos importantes en un sistema y elevar así características importantes como la alta disponibilidad o un mayor rendimiento.

- **Santiago Mancera Arturo Samuel**

Los aspectos a destacar de esta segunda práctica son, en primer lugar, los algoritmos utilizados. Línea base, Mínimos cuadrados y Holt Winters resultan mecanismos muy interesantes para poder administrar la red y proporcionar certeza sobre el comportamiento futuro con base en algoritmos matemáticos relativamente simples. Sin embargo, la parte más complicada fue la de la implementación con RRD tool, ya que existen diversos parámetros en la creación de las bases que pueden resultar en graficaciones fuera de lugar.

No obstante, el trabajo en equipo y el apoyo mutuo sirvieron para poder solventar las dificultades que se presentarán.

Referencias y bibliografías

- [1] JUAN C. HERNÁNDEZ M., *Método Holt Winters* (2017). Disponible en: http://rstudio-pubs-static.s3.amazonaws.com/283175_1d0898ed1b704812a4eeb29b1fdcb213.html [Consultado el 01 Nov. 2018].
- [2] OMAR MAGUIÑA G., *El Método de Pronóstico de Holt Winters* (2016). Disponible en: <https://administration21.files.wordpress.com/2017/01/pronc3b3sticos-holt-winters-omr-nov2016.pdf> [Consultado el 01 Nov. 2018].
- [3] CISCO., *White Paper de las mejores prácticas del proceso de línea de base* (2015). Disponible en: http://www.cisco.com/cisco/web/support/LA/102/1025/1025763_HAS_baseline.pdf [Consultado el 01 Nov. 2018].
- [4] NET.SNMP., *HOST-RESOURCES-MIB* (2011). Disponible en: <http://www.netsnmp.org/docs/mibs/host.html> [Consultado el 01 Nov. 2018].