Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 2

Integrantes:
Amador Nava Miguel Ángel
López Ayala Eric Alejandro
López Romero Joel

Profesora:
Pérez de Los Santos Mondragón
Tanibet

Grupo:
4CM1

Asignatura:
Administración de Servicios en Red

Índice general

1 | Introducción

1.1. Introducción

En la administración de servicios en red es necesario tener las capacidades que permitan el intercambio y procesamiento de información con el fin de poder estudiar la red para mantenerla trabajando de manera eficiente.
Para poder lograr ese funcionamiento eficiente se tiene que obtener la información del software y hardware así como sus niveles normales de funcionamiento para predecir problemas que más adelante generen problemas a la red.
Dentro de las herramientas para la predicción de la tendencia de series temporales se tienen:

- Línea base: Es es un proceso para estudiar la red a intervalos regulares para asegurarse de que la red está trabajando según lo diseñado. Es más que un solo informe que detalla la salud de la red en cierta punta a tiempo.
  Línea de base ayuda a identificar y a planear correctamente para los problemas de la limitación de los recursos críticos en la red.
  Estos problemas se pueden describir como los recursos de plano del control o recursos de plano de los datos. Los recursos de plano del control son únicos a la plataforma y a los módulos específicos dentro del dispositivo y pueden ser afectados por varios problemas incluyendo:
  - Utilización de los datos
  - Características habilitadas
  - Diseño de red
- Mínimos cuadrados: El método consiste en acercar una línea o una curva, según se escoja, lo más posible a los puntos determinados por la coordenadas [x, f(x)], que normalmente corresponden a muestras de algún experimento.
  Es importante aclarar que este método, aunque es sencillo de implantar no es del todo preciso, pero si proporciona una interpolación aceptable.
- Holt Winters: 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.
  Los modelos de suavizamiento exponencial se basan en la actualización, para cada período, de hasta tres parámetros:

- Media (modelo de suaviza miento simple).
- Media y tendencia (Holt, 1957).
- Media, tendencia y estacionalidad (modelo de Holt-Winters).

Estos modelos se conocen en la literatura como de suavizamiento exponencial de uno, dos y tres parámetros, respectivamente.

Estos tres métodos se utilizaran con el Protocolo simple de administración de red (SNMP) para obtener la información, para el almacenamiento y su posterior análisis se utilizará el método de Round-Robin para trabajar base de datos con cantidades fijas asegurando que el espacio de almacenamiento no se vea comprometido con el paso del tiempo.

2 | Contrato de Nivel de Servicio

Características tecnicas del equipo
El equipo es una DELL Inspiron 5556 en el *Cuadro 2.1* se presentan las caracteríticas del equipo.

Nombre del dispositivo	**Marca**	**Modelo**	**Capacidad**
CPU	AMD	A12-9700P	2.5 - 3.4 GHz x 4 Cores
RAM	Kingston	SODIMM DDR4	16 Gb (3.9 disponibles)
HDD	Western Digital	WDC WD10JPVX-75JC3T0	1000 Gb (194 Gb disponibles)

Cuadro 2.1: Caraterísticas

El agente es simulado en una máquina virtual con un sistema operativo Linux Ubuntu 18.04.1 LTS, RAM de 4 GB y 2 núcleos de procesador.

Los umbrales fueron definidos de acuerdo al comportamiento analizado por 2 horas en donde, por medio del promedio y la desviación se obtuvo el primer umbral **(ready)**, seguido del tercer umbral, el cual lo proporciona el fabricante y por último el segundo umbral el cual es el valor medio entre el primer y tercer umbral. A continuación se muestran los umbrales definidos:

- CPU. Primer umbral: 40, Segundo Umbral: 65 y Tercer umbral: 90
- RAM. Primer umbral: 55, Segundo Umbral: 67 y Tercer umbral: 80
- HDD. Primer umbral: 40, Segundo Umbral: 65 y Tercer umbral: 90
- Tráfico de res. Primer umbral: 30, Segundo Umbral: 60 y Tercer umbral: 90

Nota. En la sección 3 se explica de manera concisa todo el procedimiento para determinar los umbrales, además de la adquisición de datos de la base rrd y el notificado cuando algún valor supera dicho umbral.

Por lo tanto las gráficas de línea base resultantes son las siguientes:

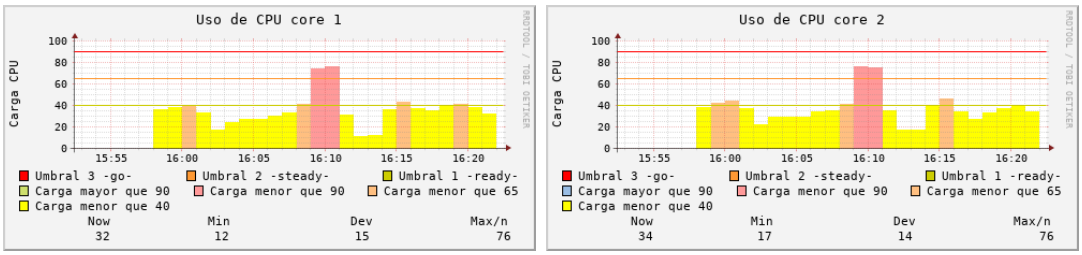


Figura 2.1: Procesador

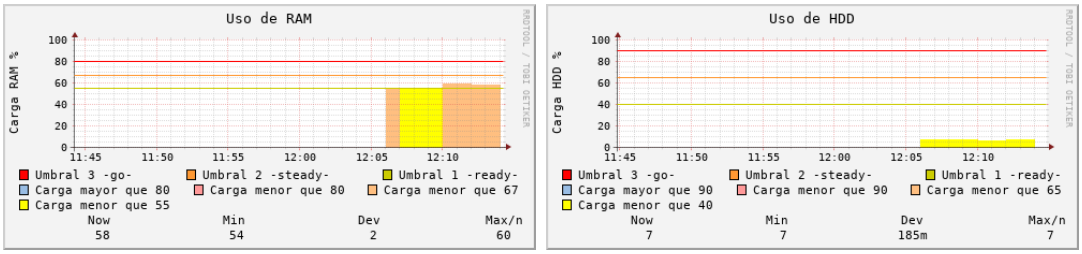


Figura 2.2: Memoria RAM y Almacenamiento del dispositivo

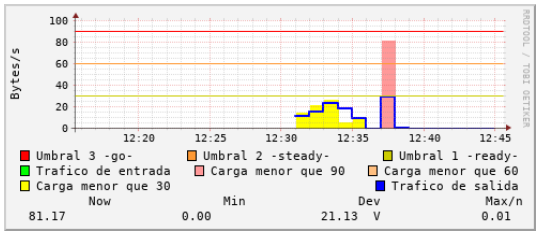


Figura 2.3: Tráfico de red

3 | Análisis y Ajuste de Rendimiento de Linea Base

3.1. Inventario de la configuración.

En el *Cuadro 3.1* se muestra el inventario de los aspectos básicos de un agente, la información presente, fue recolectada de un agente simulado en una máquina virtual con un sistema operativo Linux Ubuntu 18.04.1 LTS, RAM de 4 GB y 2 núcleos de procesador.

Nombre del dispositivo	**Versión del software (SO)**	**Tiempo de actividad del sistema**	**Fecha y hora del host**	**Número de procesos que se ejecutan en el sistema**
CPU	Ubuntu 18.04.1 LTS	3:20:27.83	02/11/18 6:52	37
RAM	Ubuntu 18.04.1 LTS	3:20:27.83	02/11/18 6:52	37
HDD	Ubuntu 18.04.1 LTS	3:20:27.83	02/11/18 6:52	37
Red	Ubuntu 18.04.1 LTS	3:20:27.83	02/11/18 6:52	37

Cuadro 3.1: Inventario

3.2. Verificar que SNMP MIB se admita en el host.

El grupo Host-Resources de la MIB define un conjunto uniforme de objetos útiles para la administración de equipos host. Las computadoras host son independientes del sistema operativo, los servicios de red o cualquier aplicación de software.

El grupo Host-Resources define objetos que son comunes en muchas arquitecturas de sistemas informáticos. Además, hay objetos en MIB-II que también proporcionan funcionalidad de administración de host. La implementación de los grupos de sistemas e interfaces es obligatoria para los implementadores de la MIB de recursos del host.[1]

De acuerdo a esta información obtenida por el RFC 1514, en este grupo de la MIB podemos obtener ciertos valores de varios objetos para poder administrar los agentes, estos valores son importantes, ya que con estos podemos obtener el comportamiento de ciertos objetos de un agente, en este caso CPU, RAM, HDD y tráfico de red. Para obtener los valores de estos objetos dentro de la MIB, se muestra a continuación su OID.

- ¿Cuál es el OID para conocer el uso del procesador?
  1.3.6.1.2.1.25.3.3.1.2

```
iso.3.6.1.2.1.25.3.3.1.2.196608 = INTEGER: 15
iso.3.6.1.2.1.25.3.3.1.2.196609 = INTEGER: 15
```

Figura 3.1

- ¿Cuál es el OID para obtener el uso de la memoria RAM?
  1.3.6.1.2.1.25.2.2
  1.3.6.1.4.1.2021.4.6 (Privado)

```
iso.3.6.1.2.1.25.2.2.0 = INTEGER: 4039540
```

Figura 3.2

- ¿Cuál es el OID para sondear el uso del almacenamiento del dispositivo?
  1.3.6.1.2.1.25.2.3.1

```
iso.3.6.1.2.1.25.2.3.1.6.1 = INTEGER: 3928020
iso.3.6.1.2.1.25.2.3.1.6.3 = INTEGER: 3928020
iso.3.6.1.2.1.25.2.3.1.6.6 = INTEGER: 178548
iso.3.6.1.2.1.25.2.3.1.6.7 = INTEGER: 1805484
iso.3.6.1.2.1.25.2.3.1.6.8 = INTEGER: 75860
iso.3.6.1.2.1.25.2.3.1.6.10 = INTEGER: 0
iso.3.6.1.2.1.25.2.3.1.6.35 = INTEGER: 405
iso.3.6.1.2.1.25.2.3.1.6.36 = INTEGER: 2072509
iso.3.6.1.2.1.25.2.3.1.6.38 = INTEGER: 10332
iso.3.6.1.2.1.25.2.3.1.6.39 = INTEGER: 1
iso.3.6.1.2.1.25.2.3.1.6.40 = INTEGER: 0
iso.3.6.1.2.1.25.2.3.1.6.80 = INTEGER: 279880
iso.3.6.1.2.1.25.2.3.1.6.81 = INTEGER: 40710
iso.3.6.1.2.1.25.2.3.1.6.82 = INTEGER: 7
iso.3.6.1.2.1.25.2.3.1.6.83 = INTEGER: 12
iso.3.6.1.2.1.25.2.3.1.6.87 = INTEGER: 405
```

Figura 3.3

- ¿Cuál es el OID para obtener el uso del tráfico de red?
  1.3.6.1.2.1.2.2.1.10.2 Tráfico de entrada
  1.3.6.1.2.1.2.2.1.16.2 Tráfico de salida

```
iso.3.6.1.2.1.2.2.1.10.2 = Counter32: 38448840
iso.3.6.1.2.1.2.2.1.16.2 = Counter32: 1435299
```

Figura 3.4

Adquisición de datos.

A continuación se presenta fragmentos de codigos sobre la adquisición de los datos de la MIB y la creación de la base rrd.

```
while 1:
    #Lectura SNMP
    carga_CPU = consultaWALKSNMP('comunidadASR','localhost','1.3.6.1.2.1.25.3.3.1.2')

    valor = "N"
    for val_CPU in carga_CPU:
        valor = valor + ":" + str(val_CPU)

    print (valor)
    rrdtool.update('trend.rrd', valor)
    rrdtool.dump('trend.rrd','trend.xml')
    time.sleep(1)
```

Figura 3.5

```
while 1:
  for i in range(num_cores):
    ret = rrdtool.graph( "imgs/LineaBaseCore" + str(i + 1) + ".png",
    "--start",str(tiempo_inicial),
    "--end",str(tiempo_final),
    "--vertical-label=Carga CPU",
    "--title=Uso de CPU core "+str(i+1),
    '--lower-limit', '0',
    '--upper-limit', '100',
    "DEF:carga" + str(i + 1) + "=trend.rrd:CPU" + str(i + 1) + "load:AVERAGE",

    #------------------------------Umbrales---------------------------------------------------------------------
    "CDEF:umbral"+ str(umbral1P) +"=carga" + str(i + 1) + ","+ str(umbral1P) +",GT,0,carga" + str(i + 1) + ",IF",
    "CDEF:umbral"+ str(umbral2P) +"=carga" + str(i + 1) + ","+ str(umbral2P) +",GT,"+ str(umbral2P) +",carga" + str(i + 1) + ",IF",
    "CDEF:umbral"+ str(umbral3P) +"=carga" + str(i + 1) + ","+ str(umbral3P) +",GT,"+ str(umbral2P) +",carga" + str(i + 1) + ",IF",
    #-----------------------------------------------------------------------------------------------------------

    #-----------Definicion de valores-----------
    "VDEF:CPUlast=carga" + str(i + 1) + ",LAST",
    "VDEF:CPUmin=carga" + str(i + 1) + ",MINIMUM",
    "VDEF:CPUdev=carga" + str(i + 1) + ",STDEV",
    "VDEF:CPUmax=carga" + str(i + 1) + ",MAXIMUM",
    #-------------------------------------------

    "AREA:carga" + str(i + 1) + color_hex[i] + ":>90",
    #--------------Impresion areas---------------------------
    "AREA:umbral"+ str(umbral3P) +"#FF9999:Carga menor que 90",
    "AREA:umbral"+ str(umbral2P) +"#FFBF80:Carga menor que 60",
    "AREA:umbral"+ str(umbral1P) +"#FFFF00:Carga menor que 30",
    #--------------------------------------------------------

    #---------------LineasBase---------------
    "HRULE:"+ str(umbral3P) +"#FF0000:Umbral 3",
    "HRULE:"+ str(umbral2P) +"#FF9933:Umbral 2",
    "HRULE:"+ str(umbral1P) +"#CCCC00:Umbral 1",
    #----------------------------------------
```

Figura 3.6

```
  "COMMENT:        Now            Min            Dev            Max/n",
  "GPRINT:CPUlast:%12.0lf%s",
  "PRINT:CPUlast:%12.0lf%s",
  "GPRINT:CPUmin:%10.0lf%s",
  "GPRINT:CPUdev:%13.0lf%s",
  "GPRINT:CPUmax:%13.0lf%s")

   print (ret[i])
  res1 = ''.join(ret[2])
  valorLast = int(res1)
  print ("core "+str(i)+": ",valorLast)
```

Figura 3.7

3.3. Consultar y registrar objetos MIB del SNMP específicos del HOST.

Para obtener los datos del uso de CPU, RAM, HDD y tráfico de red, se implementaron 4 scripts en python para realizar la adquisición de estos datos, además de crear la base rrd para posteriormente crear las gráficas. Cabe destacar que para ciertos valores de estos objetos de la MIB (RAM, HDD) devuelven cantidades que especifican en el RFC (Kb), pero para tener mayor comodidad se realizó una conversión a porcentaje, para ello fue necesario obtener el valor total de almacenamiento de estos objetos.
A continuación se muestra la obtención de dichos valores.

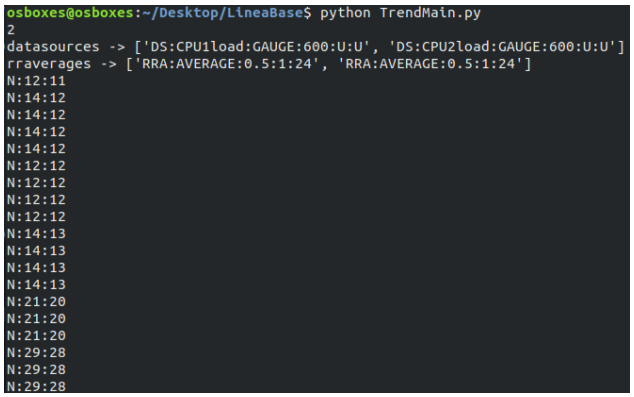


Figura 3.8: Procesador

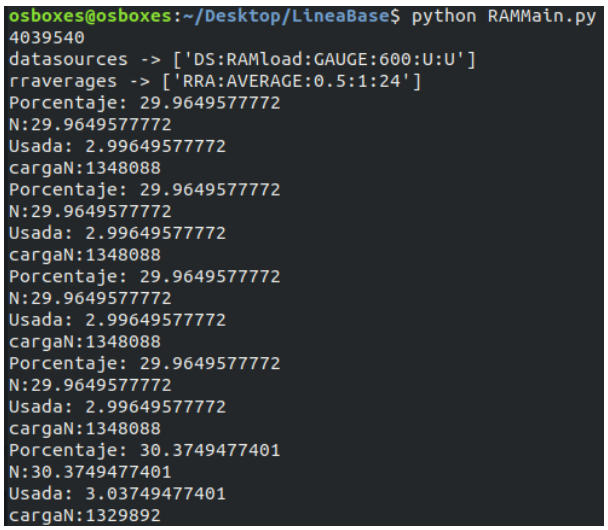


Figura 3.9: Memoria RAM

```
osboxes@osboxes:~/Desktop/LineaBase$ python HHDMain.py
154013402
datasources -> ['DS:HDDload:GAUGE:600:U:U']
rraverages -> ['RRA:AVERAGE:0.5:1:24']
11963108
Porcentaje: 7.76757596719
N:7.76757596719
Usada: 0.0776757596719
cargaN:11963108
11963108
Porcentaje: 7.76757596719
N:7.76757596719
Usada: 0.0776757596719
cargaN:11963108
11963108
Porcentaje: 7.76757596719
N:7.76757596719
Usada: 0.0776757596719
cargaN:11963108
11963108
Porcentaje: 7.76757596719
N:7.76757596719
Usada: 0.0776757596719
cargaN:11963108
```

Figura 3.10: Almacenamiento del dispositivo

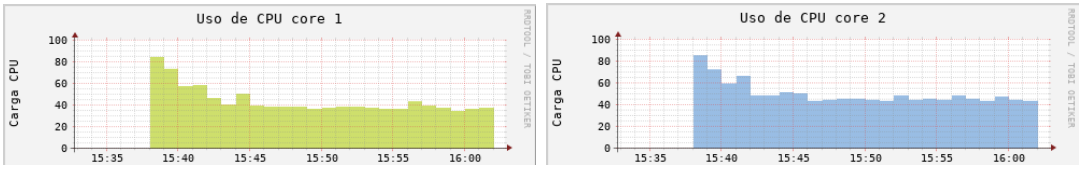
```
osboxes@osboxes:~/Desktop/LineaBase$ python TraficMain.py
N:47737193:2126937
N:47737193:2126937
N:47737193:2126937
N:47737193:2126937
N:47737668:2127710
N:47737668:2127710
N:47737668:2127710
N:47737668:2127710
N:47737668:2127710
N:47737668:2129064
N:47738390:2129064
N:47738390:2129471
N:47738620:2129471
N:47738620:2129471
N:47738620:2129471
N:47738620:2129471
N:47738620:2129471
N:47738620:2129471
N:47738620:2129471
```

Figura 3.11: Tráfico de red

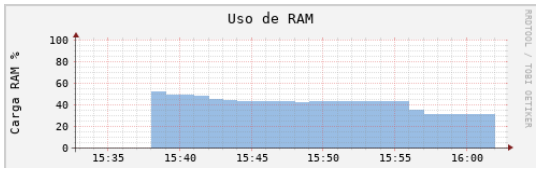
3.4. Analice los datos para determinar los umbrales.

Umbral 1: Para determinar este umbral, se analizó aproximadamente por 2 horas a cada objeto de la MIB(CPU, RAM, HDD y tráfico), de toda esta colección de valores se sacó un promedio y se le sumo un 5 % para que este no esté siempre en el umbral 1, a continuación se muestran estos promedios.

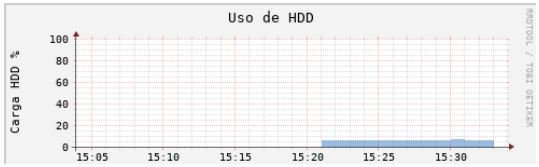
- Procesador: 40 %



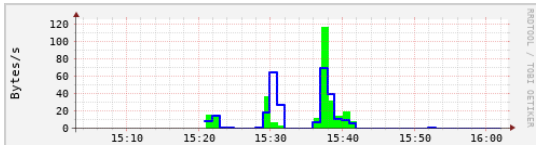
- Memoria RAM: 55 %



- HDD: 40 %



- Tráfico de red: 30 %



Umbral 3: Este umbral lo define el fabricante, el agente a monitorizar es una DELL Inspiron 5556 con CPU AMD A12-9700P, 16 GB RAM, 1Tb HDD. Investigando los datos que ofrece el fabricante, se llegó a qué:

- Procesador: El uso de procesador es muy variante, por ello el fabricante AMD, nos recomienda que sea un 90 % el uso máximo, ya que el 10 % sobrante es como un respaldo por si llegara a ocurrir algún error mientras este está a un 90 %.

- Memoria RAM: La memoria RAM es Kingston, este fabricante nos recomienda un 80 % como uso máximo, al igual que AMD, el 20 % sobrante es un respaldo por si llegara a ocurrir algún error y evitar que este afecte al funcionamiento del agente.

- HDD: el fabricante del HDD es Western Digital y este recomienda un 90 % aunque no especifica porque el 10 % de sobra.

- Tráfico de red: El dispositivo es Qualcomm QCA9377, Qualcomm no da como tal una recomendación del uso de su dispositivo, ya que este tiende a un comportamiento de altas y bajas, es decir un comportamiento de cierta forma senoidal, ya que recibe cierta cantidad de bytes y luego regresa a una cantidad mínima, como tal no se mantiene a diferencia de la memoria RAM o CPU. Investigando en internet, llegué a la conclusión de un 90 %.

Umbral 2: El umbral 2 es el valor medio entre el umbral 1 y el umbral 2, por lo tanto tenemos lo siguiente:

- Procesador: 65 %

- Memoria RAM: 67 %

- HDD: 65 %

- Tráfico de red: 60 %

3.5. Problemas inmediatos identificados arreglo.

Una vez definidos los umbrales, estos los especificamos al momento de crear la gráfica de cada objeto de la MIB, para ello definimos una colección con un IF para cada umbral, es decir, si un valor se pasa un umbral, estos datos de guardan dentro de una colección en rrdtool y en la gráfica se distingue por un color diferente. A continuación se muestran las gráficas.

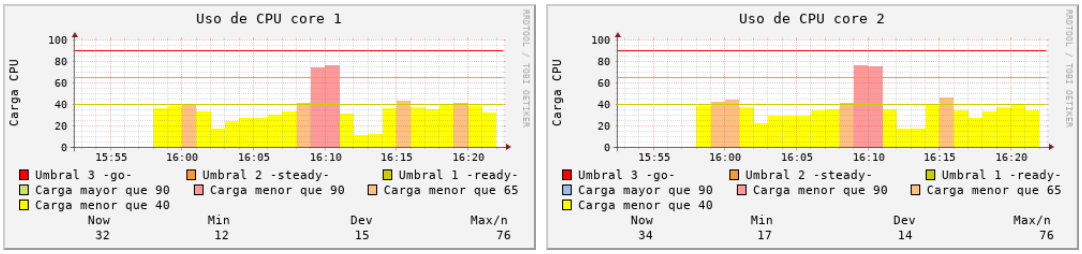


Figura 3.12: Procesador

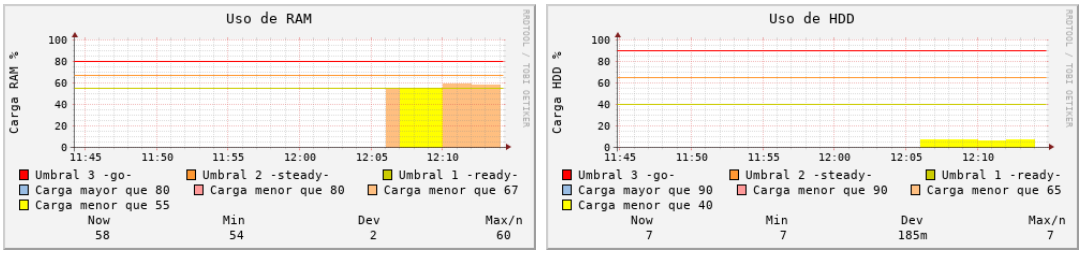


Figura 3.13: Memoria RAM y Almacenamiento del dispositivo

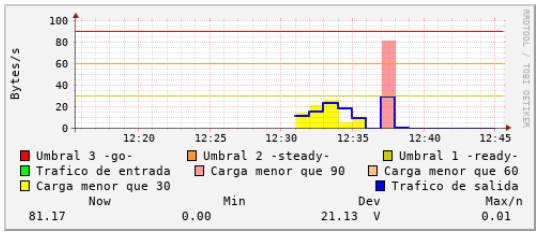


Figura 3.14: Tráfico de red

Como se observa, cada gráfica tiene sus umbrales definidos, además se especifica cuando se sobrepasa algún umbral. Una vez ya identificados estos umbrales dentro de la gráfica y en la colección de datos, se planea implementar una notificación para informar al administrador de que dicho objeto dentro del agente está pasando algún umbral.

En el siguiente código se muestra como se determina los umbrales para gráficarlos, además el código para verificar cuando se pasa algun umbral y posteriormente el envío de notificaciones.

```
#------------------------------Umbrales-----------------------------------
"CDEF:umbral"+ str(umbral1P) +"=carga" + str(i + 1) + ","+ str(umbral1P) +"
"CDEF:umbral"+ str(umbral2P) +"=carga" + str(i + 1) + ","+ str(umbral2P) +"
"CDEF:umbral"+ str(umbral3P) +"=carga" + str(i + 1) + ","+ str(umbral3P) +"
#------------------------------------------------------------------------
```

```
if bandera == 1:
    bandera == 0
    if (valorLast >= umbral1P and valorLast < umbral2P):
        if (bandera == 1):
            bandera = 1
        else:
            print ("correo umbral 1")              #Enviamos correo
            sendEmail("Umbral 1 arrevasado -ready-", "LineaBaseCore" + str(
            bandera = 1
            return Processor(bandera)
    else:
        if bandera == 1:
            bandera == 0
        if (valorLast >= umbral2P and valorLast < umbral3P):
            if (bandera == 1):
                bandera = 1
            else:
                print ("correo umbral 2")          #Enviamos correo
                sendEmail("Umbral 2 arrevasado -steady-", "LineaBaseCore" +
                bandera = 1
                return Processor(bandera)
        else:
            if bandera == 1:
                bandera == 0
            if (valorLast >= umbral3P):
                if (bandera == 1):
                    bandera = 1
                else:
                    print ("correo umbral 3")      #Enviamos correo
                    sendEmail("Umbral 3 arrevasado -go-", "LineaBaseCore" +
                    bandera = 1
                    return Processor(bandera)
```

4 | Supervisión del Rendimiento, Medición e Informes

4.1. Supervisión del rendimiento, medición e informes.

Como se mencionó anteriormente, se implementara una notificación. En esta notificación se usa la librería smtplib para enviar correos por medio de smtp. Antes de enviar alguna notificación se debe corroborar que el dato este sobrepasando algún umbral, además de enviar solo un correo. Es por esto que se realizó un método para corroborar que el último valor que se está obteniendo de la base rrd, está dentro de algún umbral y con ello enviar la notificación al administrador. Para que solo se envié una notificación, se usaron banderas como auxiliares.
A continuación se muestra el código de la notificación.

```
1  import smtplib
2  from email.mime.image import MIMEImage
3  from email.mime.multipart import MIMEMultipart
4  import time
5
6  #URL de la ubicacion de las graficas
7  pngpath = '/home/osboxes/Desktop/LineaBase/imgs/'
8
9  #---------------------Envio de correos
10 #Metodo para enviar los correos, recibimos asunto e imagen
11 def sendEmail(subject, imag):
12
13     msg = MIMEMultipart()  #Iniciamos el cuerpo del correo
14
15     msg['Subject'] = subject
16
17     #------------Credenciales--------------
18     msg['From'] = 'escomcec@gmail.com'
19     msg['To'] = 'jllpzrmr7@gmail.com'
20     password = "cec12345"
21     #--------------------------------------
22
```

```
23      fp = open(pngpath+imag, 'rb')  #Abrimos imagen
24
25      msg.attach(MIMEImage(fp.read())) #Anexamos la imagen al correo
26
27      #Iniciamos una conexion con el servidor smtp de Google
28      s = smtplib.SMTP('smtp.gmail.com: 587')
29      s.starttls()
30
31      #Login a la cuenta con las credenciales
32      s.login(msg['From'], password)
33
34      #Enviamos el correo por el servidor
35      s.sendmail(msg['From'], msg['To'], msg.as_string())
36
37      #Cerramos la conexion con el servidor smtp del Google
38      s.quit()
39
40      print ("successfully sent email to %s:" % (msg['To']))
```

Las siguientes figuras muestran la envidencia del envío de correos con la respectiva gráfica.

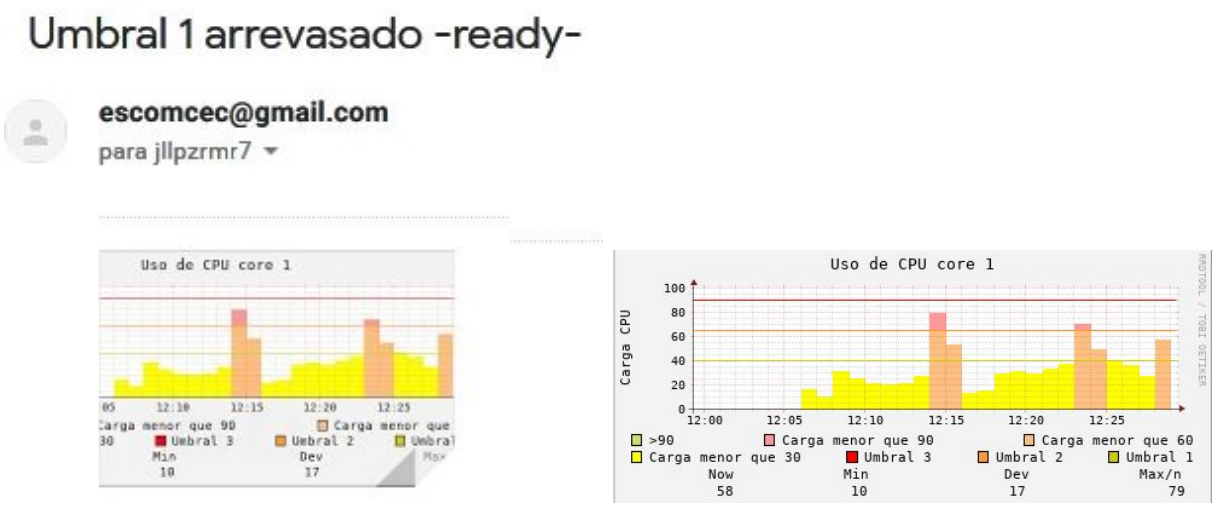


Figura 4.1: Umbral 1

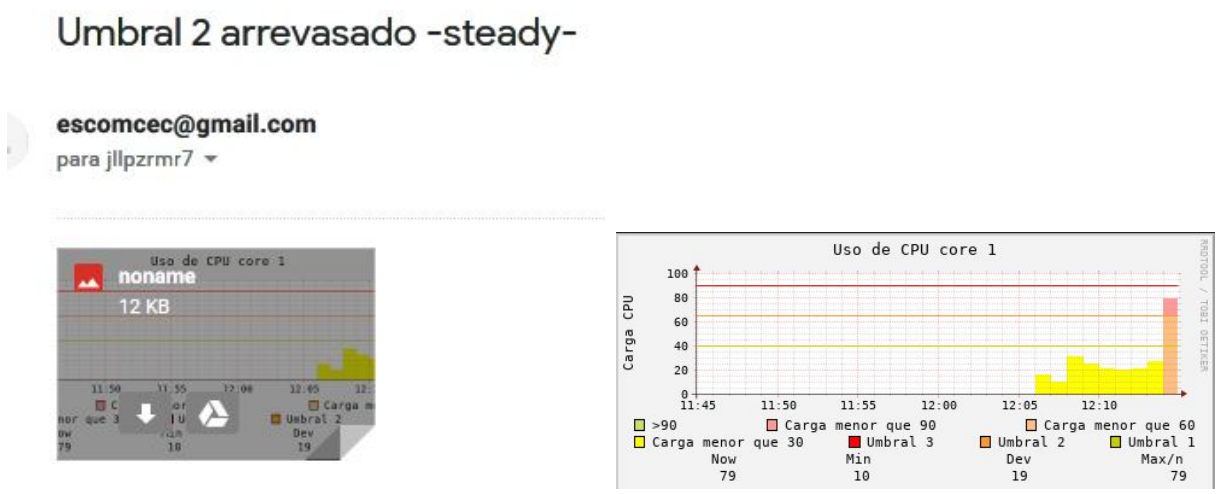


Figura 4.2: Umbral 2

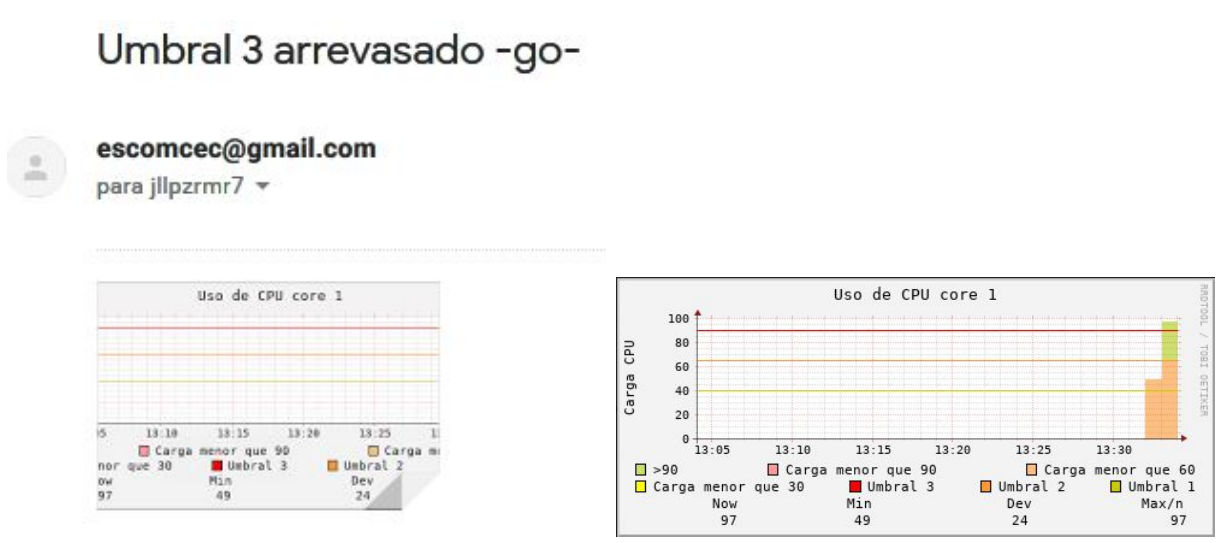


Figura 4.3: Umbral 3

5 | Detección de Comportamiento Anómalo

5.1. Detección de comportamiento anómalo al monitorizar una red.

5.1.1. Predicción de la tendencia de series temporales lineales.

Mínimos Cuadrados

El método de mínimos cuadrados se aplica para ajustar rectas a una serie de datos presentados como puntos en el plano.
Dado un conjunto de pares ordenados $(x_1, y_1), (x_2, y_3), ..., (x_n, y_n)$ aplicando mínimos cuadrados podemos esperar una relación lineal:

$$y = mx + b \tag{5.1}$$

El método de mínimos cuadrados nos proporciona un criterio mediante el cual obtenemos la mejor recta que representa a los puntos dados.
Método:
Se calcula la media de los valores de X y Y.

$$\bar{X} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{5.2}$$

$$\bar{Y} = \frac{\sum_{i=1}^{n} y_i}{n} \tag{5.3}$$

Se obtiene la pendiente de la línea:

$$m = \frac{\sum_{i=1}^{n}(x_i - \bar{X})(y_i - \bar{Y})}{\sum_{i=1}^{n}(x_i - \bar{X})^2} \tag{5.4}$$

Para la intersección en y de la línea:

$$b = \bar{Y} - m\bar{X} \tag{5.5}$$

Con base en la teoría de mínimos cuadrados se puede implementar una herramienta para monitorear el comportamiento de la red. El siguiente código muestra la implementaciíon en python con la biblioteca rrdtool.

Implementación de mínimos cuadrados

```
import sys
import rrdtool
import time

nombreRRD = "source3"
proyeccion = 5000
etiqueta_vertical = "Carga CPU"
etiqueta_vertical_porcentaje= "Uso de CPU (%)"
titulo_grafica = "Uso de CPU"
titulo_area = "CPU load"
tope = 90

def graficar(cpu, nombre):
    tiempo_final = int(rrdtool.last(str(nombreRRD) + ".rrd"))
    tiempo_inicial = 1539657633
    ret = rrdtool.graph(nombre,
        "--start", str(tiempo_inicial),
        "--end", str(tiempo_final + proyeccion),
        "--vertical-label="+ str(etiqueta_vertical),
        "--title="+str(titulo_grafica),
        "--color", "ARROW#009900",
        '--vertical-label', etiqueta_vertical_porcentaje,
        '--lower-limit', '0',
        '--upper-limit', '100',
        "DEF:carga="+str(nombreRRD)+".rrd:"+str(cpu)+":AVERAGE",
        "AREA:carga#00FF00:"+str(titulo_area),
        "LINE1:"+str(tope)+"#FF0000",
        "AREA:5#ff000022:stack",
        "VDEF:last=carga,LAST",
        "VDEF:min=carga,MINIMUM",
        "VDEF:avg=carga,AVERAGE",
        "VDEF:max=carga,MAXIMUM",
        "COMMENT:          Now              Min                  Avg
Max",
        "GPRINT:last:%12.0lf%s",
        "GPRINT:min:%10.0lf%s",
        "GPRINT:avg:%13.0lf%s",
        "GPRINT:max:%13.0lf%s",
```

```
38          "VDEF:a=carga,LSLSLOPE",
39          "VDEF:b=carga,LSLINT",
40          'CDEF:avg2=carga,POP,a,COUNT,*,b,+',
41          'VDEF:maxColect=avg2,MAXIMUM',
42          'CDEF:result=avg2,'+str(tope)+',GE,avg2,maxColect,IF',
43          'VDEF:respuesta=result,MINIMUM',
44          'PRINT:respuesta:"%c":strftime',
45          "LINE2:avg2#FFBB00:Minimos Cuadrados",
46          "COMMENT:Interseccion ",
47          'GPRINT:respuesta:"%c":strftime')
48
49  graficar("CPUload","cpu1.png")
```

Variables en el código

Dentro del código se definieron las variables:
nombreRRD: Representa el nombre del archivo rrd donde se almacenó la información.
proyeccion: Una vez que con mínimos cuadrados obtenemos la función de la recta, este es el tiempo que se proyectará la gráfica después de terminar de leer los daos de la base de datos.
tope: Es el valor crítico donde se intersectará la proyección.
tiempo inicial: Fecha de inicio del análisis para su posterior graficación (en tiempo UNIX).
La función:
graficar(cpu, nombre) Grafica la proyección de mínimos cuadrados y muestra la fecha en que se llegará al punto crítico.
Donde:
cpu: Es el nombre del data source del archivo rrd.
nombre: Es el nombre que recivirá la gráfica.
Esta función debe ser llamada para cada nucleo que tenga un equipo en la red.

Parámetros de RRDtool

Los parametros más importantes en la implementación de mínimos cuadros son:
graph: Crea un gráfico a partir de los datos almacenados en uno o varios RRDs, también permite extraer datos.
PRINT: Regresa el valor de una varible a la salida estándar.
GPRINT: Imprime en el gráfico la variable indicada.
VDEF: Define una varible.
CDEF: Define una colección de datos.
Definimos en la línea 38 la variable **a** , la cual será la pendiente de la recta

```
1     "VDEF:a=carga,LSLSLOPE",
```

En la línea 39 difinimos la variable **b** la cual será la intersección en el eje y.

```
1     "VDEF:b=carga,LSLINT",
```

En la línea 40 generamos la recta, definiendo una colección de datos llamada avg2, donde carga son los datos almacenados en el archivo rrd que fueron extraídos de acuerdo a la fuente de datos

(DS: data source) y recorridos uno por uno y trabajados con la función de la recta.

```
1         'CDEF:avg2=carga,POP,a,COUNT,*,b,+',
```

De la línea 41 a la 43, definimos una variable que almacena el valor máximo de avg2 (incluye los valores de la proyección), para después generar una sentencia booleana que almacena en una colección de datos (result) todos los valores que son mayores o igual (GE) a *tope* para finalmente tomar el valor mínimo de la colección (respueta) que será el valor donde se intersectará la proyección.

```
1       'VDEF:maxColect=avg2,MAXIMUM',
2       'CDEF:result=avg2,'+str(tope)+',GE,avg2,maxColect,IF',
3       'VDEF:respuesta=result,MINIMUM',
```

Resultados de la implementación

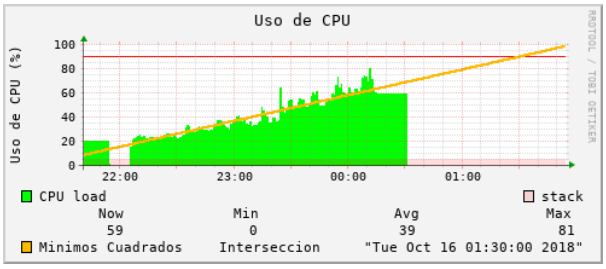


Figura 5.1: Ejemplo de predicción con mínimos cuadrados

5.1.2. Predicción de la tendencia de series temporales no lineales.

Predicción mediante el método de Holt-Winters

El suavizamiento exponencial triple, también conocido como el método de **Holt Winters**, es uno de los muchos algoritmos que se pueden usar para pronosticar puntos de datos en una serie, siempre y cuando esta serie sea *estacional* en algún periodo.

Para poder comprender e implementar el método de Holt-Winters para la predicción de la tendencia de series temporales no lineales, fue necesario implementar primero métodos de predicción más simples. A continuación se mencionarán los métodos implementados y las contribuciones de cada uno de ellos para poder implementar el método de Holt-Winters.

- **Método Naive:** Es el método más primitivo para realizar una predicción. La premisa de este método es que el valor del punto esperado será igual al último punto observado.

$$y_{x+1} = y_x \tag{5.6}$$

- **Promedio Simple:** Un método un poco menos primitivo es realizar el promedio de todos los puntos previamente observados. La premisa de este método es que el valor del punto esperado será (o se aproximará) al promedio calculado.

$$y_{x+1} = \frac{1}{n}\sum_{i=1}^{x} y_i \tag{5.7}$$

A continuación se muestra la implmentación de este método codificado en el lenguaje de programación pyhton:

```python
def average(series):
    return float(sum(series))/len(series)

# Given the above series, the average is:
# >>> average(series)
# 10.285714285714286
```

Figura 5.2: Código implementando el método de promedio simple.

- **Promedio Móvil:** Es una mejora sobre el método de promedio simple, en el cual el promedio únicamente serán los últimos n puntos de la serie obtenidos mediante el uso de una *"ventana deslizante"*. La premisa de este método es darle valor únicamente a los últimos

puntos observados (ya que son los que realmente nos interesan).

$$y_{x+1} = \frac{1}{n} \sum_{i=x-n}^{x} y_i \tag{5.8}$$

A continuación se muestra la implmentación de este método codificado en el lenguaje de programación pyhton:

```python
# moving average using n last points
def moving_average(series, n):
    return average(series[-n:])

# >>> moving_average(series, 3)
# 11.333333333333334
# >>> moving_average(series, 4)
# 11.75
```

Figura 5.3: Código implementando el método de promedio móvil.

- **Promedio Móvil Ponderado:** Es un promedio móvil donde dentro de la ventana deslizante a los valores se les asignan distintos pesos. La premisa de este método es que comunmente a los puntos recientes son a los que se les da una importancia mayor.

$$y_{x+1} = \frac{1}{n} \sum_{i=x-n}^{x} y_i \cdot z, z \in \mathbb{R}^+ \tag{5.9}$$

A continuación se muestra la implmentación de este método codificado en el lenguaje de programación pyhton:

```python
# weighted average, weights is a list of weights
def weighted_average(series, weights):
    result = 0.0
    weights.reverse()
    for n in range(len(weights)):
        result += series[-n-1] * weights[n]
    return result

# >>> weights = [0.1, 0.2, 0.3, 0.4]
# >>> weighted_average(series, weights)
# 11.5
```

Figura 5.4: Código implementando el método de promedio móvil ponderado.

La gráfica 5.6 muestra una comparación entre los métodos anteriormente mencionados dada la siguiente serie:

```
1  series = [3,10,12,13,12,10,12]
```

Figura 5.5: Vector de serie de prueba.

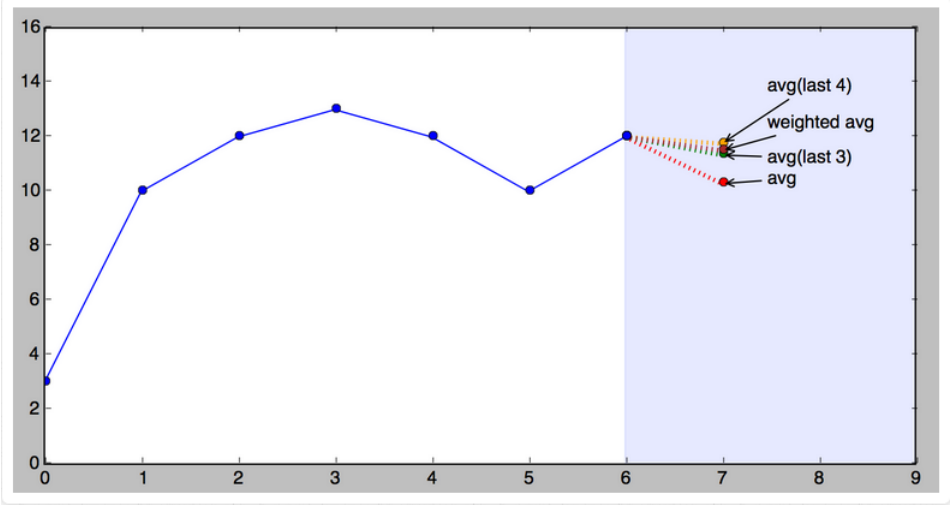


Figura 5.6: Comparación entre métodos simples.

- **Suavizamiento Exponencial Simple:** Similar al promedio móvil ponderado, ahora se considera a todos los puntos de la serie y se les asigna un valor decreciente conforme el punto sea más antiguo, tal que la asignación de los pesos decae de forma uniforme. La premisa de este método es darle un valor exponencialmente mayor a los puntos recién observados de la serie, y establecer un mecanismo de forma que la serie posea algo llamado *memory decay rate*, el cual dictamina que tan rápido una serie *olvida* algún valor.

$$y_{\mathrm{x}} = \alpha \cdot y_{\mathrm{x}} + (1 - \alpha) \cdot y_{\mathrm{x\text{-}1}}, \alpha \in (0, 1) \tag{5.10}$$

A continuación se muestra la implmentación de este método codificado en el lenguaje de programación pyhton:

```python
# given a series and alpha, return series of smoothed points
def exponential_smoothing(series, alpha):
    result = [series[0]] # first value is same as series
    for n in range(1, len(series)):
        result.append(alpha * series[n] + (1 - alpha) * result[n-1])
    return result

# >>> exponential_smoothing(series, 0.1)
# [3, 3.7, 4.53, 5.377, 6.0393, 6.43537, 6.991833]
# >>> exponential_smoothing(series, 0.9)
# [3, 9.3, 11.73, 12.873000000000001, 12.0873, 10.20873, 11.820873]
```

Figura 5.7: Código implementando el método de suavizamiento exponencial simple.

La gráfica 5.8 muestra una comparación entre los métodos anteriormente mencionados dada la siguiente serie:

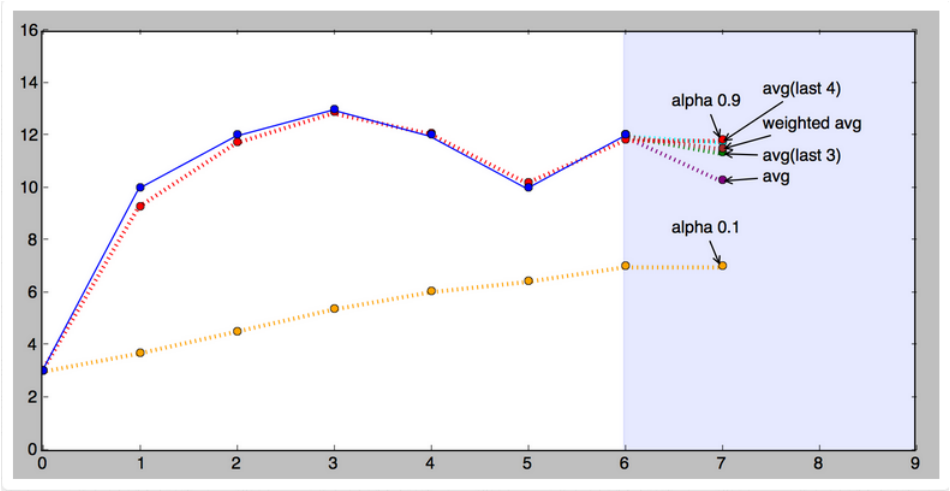


Figura 5.8: Comparación entre métodos simples vs Suavizamiento Exponencial Simple.

- **Suavizamiento Exponencial Doble:** En este método se consideran 2 componentes para la serie: nivel y tendencia, como se explicó anteriormente el suavizamiento exponencial simple nos permite predecir el nivel, ahora también podemos presecir la tendencia. La premisa de esté método es que la predicción no sea solo con base al nivel de los puntos calculados, sino también a la pendiente formada entre ellos.

$$l_x = \alpha \cdot y_x + (1 - \alpha)(l_{x\text{-}1} + b_{x\text{-}1}) - Level \tag{5.11}$$

$$b_x = \beta(l_x - l_{x\text{-}1}) + (1 - \beta)(b_{x\text{-}1}) - Trend \tag{5.12}$$

$$y_{x+1} = l_x + b_x - Forecast \tag{5.13}$$

```python
# given a series and alpha, return series of smoothed points
def double_exponential_smoothing(series, alpha, beta):
    result = [series[0]]
    for n in range(1, len(series)+1):
        if n == 1:
            level, trend = series[0], series[1] - series[0]
        if n >= len(series): # we are forecasting
          value = result[-1]
        else:
          value = series[n]
        last_level, level = level, alpha*value + (1-alpha)*(level+trend)
        trend = beta*(level-last_level) + (1-beta)*trend
        result.append(level+trend)
    return result

# >>> double_exponential_smoothing(series, alpha=0.9, beta=0.9)
# [3, 17.0, 15.45, 14.210500000000001, 11.396044999999999, 8.183803049999998, 12.753698384
```

Figura 5.9: Código implementando el método de suavizamiento exponencial doble.

La gráfica 5.8 muestra una comparación entre los métodos anteriormente mencionados dada la siguiente serie:

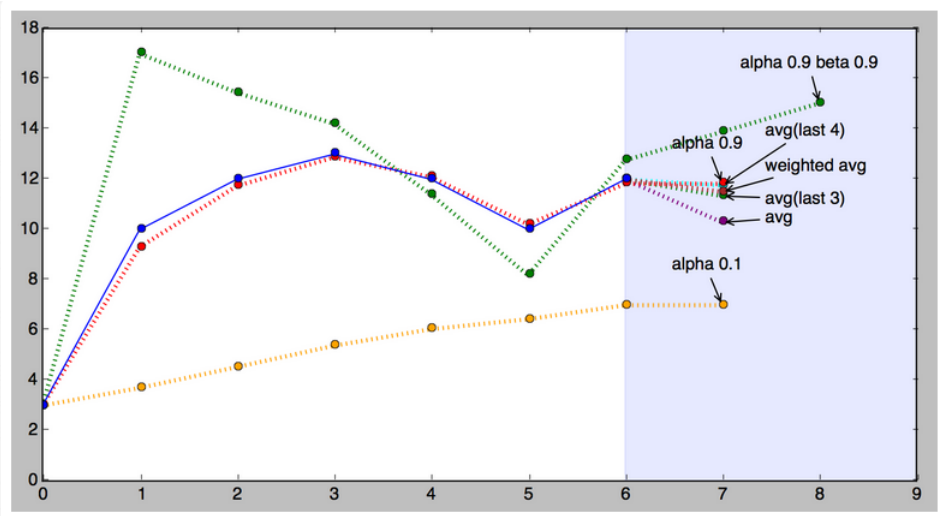


Figura 5.10: Comparación entre métodos anteriores vs Suavizamiento Exponencial Doble.

La idea detrás del suavizamiento exponencial triple es aplicar el suavizamiento a los componentes de temporada en adición al nivel y a la tendencia.

El suavizamiento se aplica a lo largo de las temporadas, e.g: un punto de la 3ra temporada es suavizado con el 3er punto de la temporada anterior y este a su vez será suavizada con el de la temporada pasada, de esta forma se agrega la 4ta ecuación que describe el suavizamiento entre temporadas:

$$l_{\text{x}} = \alpha(y_{\text{x}} - s_{\text{x-L}}) + (1-\alpha)(l_{\text{x-1}} + b_{\text{x-1}}) - \textit{Level} \tag{5.14}$$

$$b_{\text{x}} = \beta(l_{\text{x}} - l_{\text{x-1}}) + (1-\beta)(b_{\text{x-1}}) - \textit{Trend} \tag{5.15}$$

$$s_{\text{x}} = \gamma(y_{\text{x}} - l_{\text{x}}) + (1-\gamma)s_{\text{x-L}} - \textit{Seasonal} \tag{5.16}$$

$$y_{\text{x+m}} = l_{\text{x}} + mb_{\text{x}} + s_{\text{x-L+1+(m-1)modL}} - \textit{Forecast} \tag{5.17}$$

Implementación del método de Holt-Winters

A continuación se muestra la implmentación de este método codificado en el lenguaje de programación pyhton:

1. Definimos un vector de valores de prueba para la serie, tal y como se muestra en la imagen 5.11 **Vector de Serie de Prueba**.

```
series = [30,21,29,31,40,48,53,47,37,39,31,29,17,9,20,24,27,35,41,38,
          27,31,27,26,21,13,21,18,33,35,40,36,22,24,21,20,17,14,17,19,
          26,29,40,31,20,24,18,26,17,9,17,21,28,32,46,33,23,28,22,27,
          18,8,17,21,31,34,44,38,31,30,26,32]
```

Figura 5.11: Vector de Serie de Prueba.

2. Al gráficar la serie se observará como la imagen 5.12 **Gráfica de Serie de Prueba**.

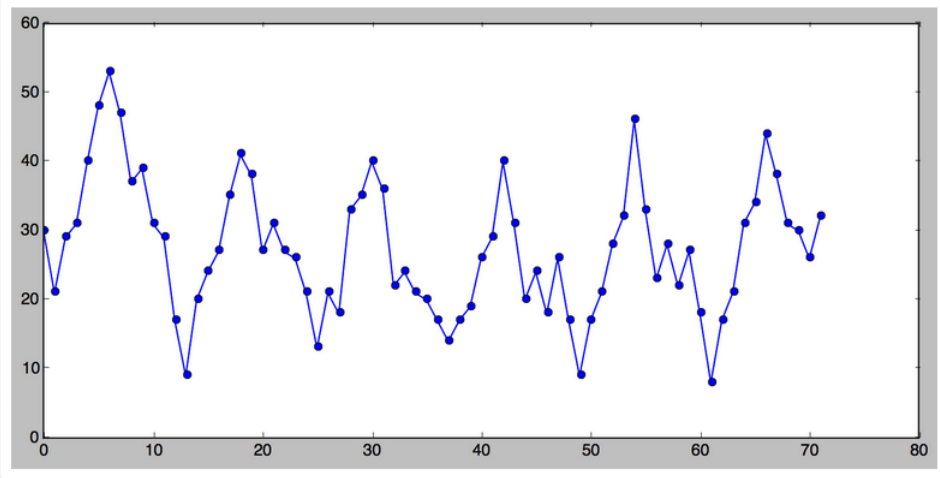


Figura 5.12: Gráfica de Serie de Prueba.

3. Se define el método para calcular la tendencia inicial a través de las temporadas, tal y como se muestra en el código 5.13 **Calculo de la Tendencia Inicial**.

```
def initial_trend(series, slen):
    sum = 0.0
    for i in range(slen):
        sum += float(series[i+slen] - series[i]) / slen
    return sum / slen

# >>> initial_trend(series, 12)
# -0.7847222222222222
```

Figura 5.13: Código implementando el cálculo de la tendencia inicial.

4. Se define el método para calcular los valores de los componentes iniciales, tal y como se muestra en el código 5.14 **Calculo de los Componentes de la Temporada Inicial**.

```python
def initial_seasonal_components(series, slen):
    seasonals = {}
    season_averages = []
    n_seasons = int(len(series)/slen)
    # compute season averages
    for j in range(n_seasons):
        season_averages.append(sum(series[slen*j:slen*j+slen])/float(slen))
    # compute initial values
    for i in range(slen):
        sum_of_vals_over_avg = 0.0
        for j in range(n_seasons):
            sum_of_vals_over_avg += series[slen*j+i]-season_averages[j]
        seasonals[i] = sum_of_vals_over_avg/n_seasons
    return seasonals

# >>> initial_seasonal_components(series, 12)
# {0: -7.4305555555555545, 1: -15.097222222222221, 2: -7.263888888888888, 3: -5.0972222222
```

Figura 5.14: Código implementando el cálculo de los componentes de la temporada inicial.

5. Implementamos el algoritmo de Holt-Winters haciendo uso de las ecuaciones previamente vistas y de los métodos anteriores para el cálculo de la tendencia inicial y los componentes de temporada iniciales, tal y como se muestra en el código 5.15 **Código implementando el algoritmo Holt-Winters**.

```python
def triple_exponential_smoothing(series, slen, alpha, beta, gamma, n_preds):
    result = []
    seasonals = initial_seasonal_components(series, slen)
    for i in range(len(series)+n_preds):
        if i == 0: # initial values
            smooth = series[0]
            trend = initial_trend(series, slen)
            result.append(series[0])
            continue
        if i >= len(series): # we are forecasting
            m = i - len(series) + 1
            result.append((smooth + m*trend) + seasonals[i%slen])
        else:
            val = series[i]
            last_smooth, smooth = smooth, alpha*(val-seasonals[i%slen]) + (1-alpha)*(smoot
            trend = beta * (smooth-last_smooth) + (1-beta)*trend
            seasonals[i%slen] = gamma*(val-smooth) + (1-gamma)*seasonals[i%slen]
            result.append(smooth+trend+seasonals[i%slen])
    return result

# # forecast 24 points (i.e. two seasons)
# >>> triple_exponential_smoothing(series, 12, 0.716, 0.029, 0.993, 24)
# [30, 20.34449316666667, 28.410051892109554, 30.438122252647577, 39.466817731253066, ...
```

Figura 5.15: Código implementando el algoritmo Holt-Winters.

6. Finalmente en la imagen 5.16 **Predicción de Valores** podemos observar una gráfica, en la cual con base a la serie inicial se realizó la predicción de 24 puntos a futuro.

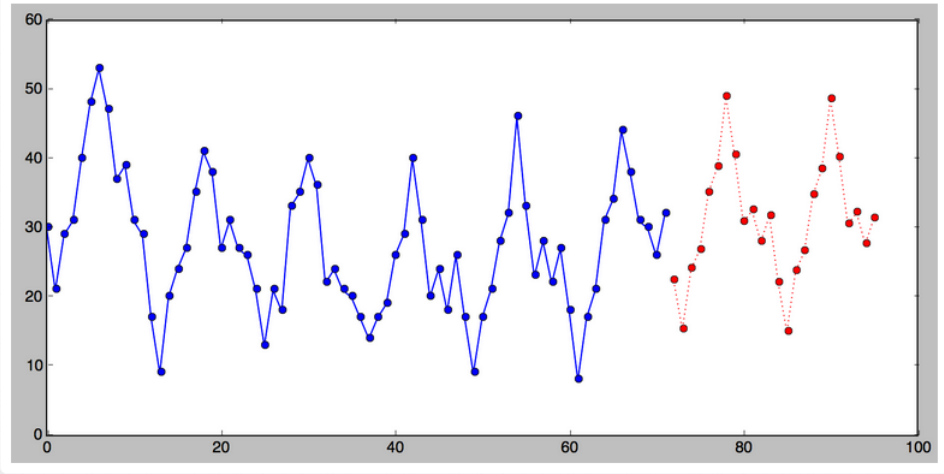


Figura 5.16: Predicción de Valores.

Uso de RRDTool para la detección de comportamiento aberrante

RRDTool nos ofrece la capacidad de realizar la detección de comportamiento aberrante en una red, para ello descompone este proceso en 3 partes:

1. Un algoritmo para poder predecir los valores de una serie de tiempo en un tiempo futuro. (Holt-Winters)
2. Una medida de desviación entre los valores predecidos y los valores observados.
3. Un mecanismo para decidir cuando un valor se desvia demasiado de los valores previstos.

Arquitectura

La base de datos round robin (RRD) es organizada en secciones secuenciales llamadas archivos round robin (RRA). Dentro cada RRA es una sección para cada una de las fuentes de datos (input) almacenados dentro del RRD. Cada RRA es definido por una función de consolidación, el cual mapea los puntos de datos primarios (PDP) en puntos de datos consildados, estos puntos son actualizados en los RRA mediante una función de forma secuencial cada cierto periodo de tiempo.

Nuevos RRA

La implementación del algoritmo de comportamiento aberrante agrega 5 nuevas *"funciones de consolidación"* en RRDTool:

1. **HWPREDICT**: Un arreglo de predicción calculado por el algoritmo Holt-Winters por cada punto de dato primario. El arreglo HWPREDICT depende de un array del tipo **SEASONAL** para poder obtener los coeficientes de temporada. El índice de este arreglo es almacenado por el parámetro **HWPREDICT RRA**.

2. **SEASONAL**: Un arreglo de coeficientes de temporada con una longitud igual al periodo de la temporada. Para cada punto de dato primario existe un coeficiente de temporada que coincide con el índice en cada ciclo de cada temporada. Para mantener el orden de la actualización de los componentes de temporada mediante suavizamiento exponencial, el **SEASONAL RRA** depende del **HWPREDICT RRA**.

3. **DEVPREDICT**: Un arreglo de predicciones de desviaciones. Esencialmente, **DEVPREDICT** copia los valores del arreglo **DEVSEASONAL**.

4. **DEVSEASONAL**: Un arreglo de desviaciones temporales. Para cada punto de dato primario, existe una desviación temporal que coincide con el índice del punto de dato primario en el ciclo de la temporada. Para poder calcular las desviaciones, **DEVSEASONAL RRA** debe ser capaz de calcular el valor predecido, por lo que depende de **HWPREDICT RRA**. El índice del HWPREDICT RRA que coincide es almacenado como parámetro.

5. **FAILURES**: Un arreglo de indicadores boleanos, un 1 indica una falla. El buffer de RRA-DS almacena los valores dentro de la ventana. Cada actualización elimina el valor más antiguo del buffer e inserta una nueva observación. Para cada actualización, el número de violaciones es recalculada. La longitud máxima de la ventana soportada por este buffer es de 9 puntos en el tiempo. Para la comparación de las desviaciones depende de **DEVSEASONAL RRA**. El índice del DEVSEASONAL RRA que coincide es almacenado como parámetro.

Creando un Archivo RRD

Lo primero que se realizó, fue crear un archivo RRD con la detección de comportamiento aberrante habilitado. Para el código a continuación se definierón de forma éxplicita la creación de los RRA de **HWPREDICT**, **SEASONAL**, **DEVPREDICT**, **DEVSEASONAL** y **FAILURES**.

```
1  #!/usr/bin/env python
2
3  import rrdtool
4  ret = rrdtool.create("predict.rrd",
5                       "--start",'N',
6                       "--step",'60',
7                       "DS:inoctets:COUNTER:600:U:U",
8                       "RRA:AVERAGE:0.5:1:2016",
9                       "RRA:HWPREDICT:1000:0.1:0.0035:288:3",
10                      "RRA:SEASONAL:288:0.1:2",
11                      "RRA:DEVSEASONAL:288:0.1:2",
12                      "RRA:DEVPREDICT:1000:4",
13                      "RRA:FAILURES:288:7:9:4")
14 if ret:
15     print rrdtool.error()
```

La declaración de los elementos cumple con la sintáxis definida a continuación:

- **RRA:HWPREDICT:array length:alpha:beta:seasonal period[:rra - num]**
  - <array length>: Es el número de predicciones que se pueden almacenar antes de cubrir la RRD; este número debe ser mayor que el periodo de la temporada. A su vez este valor también es el RRA contador de filas para DEVPREDICT RRA.
  - <alpha>: Es el paramétro de intercepción del algortimo de Holt-Winters, su valor debe estar entre el 0 y 1. Su valor debe ser el mismo que se usa para gamma.
  - <beta>: Es el parámetro de adaptación de la pendiente del algoritmo de Holt-Winters, su valor debe estar entre el 0 y 1.
  - <seasonal period[:rra - num]>: Es el número puntos de datos primarios en el periodo de la temporada. Este valor será el RRA contador de filas para los RRAs SEASONAL y DEVSEASONAL.
- **RRA:SEASONAL:period:gamma:index of HWPREDICT**
  - <period>: Es el número de puntos de datos primarios en el periodo de la temporada. Debe coincidir con el valor especificado por el argumento de <period>de HWPREDICT. Debe ser un valor entero mayor a 2.
  - <gamma>: Es la adaptación del parámetro de los coeficientes de temporada, el cual debe tener un valor entre 0 y 1.
  - <index of HWPREDICT>: Es el índice del arreglo HWPREDICT en el orden en el cual se especificarón los RRAs.
- **RRA:DEVSEASONAL:period:gamma:index of HWPREDICT**
  Los argumentos de DEVSEASONAL son los mismos que los de SEASONAL, debido a que este es una copia del arreglo anterior.
- **RRA:DEVPREDICT:period:index of DEVSEASONAL**
  - <period>: Es el número de puntos de datos primarios en el periodo de la temporada. Debe coincidir con el valor especificado por el argumento de <period>de HWPREDICT. Debe ser un valor entero mayor a 2.
  - <index of DEVSEASONAL>: Es el índice del arreglo DEVSEASONAL en el orden en el cual se especificarón los RRAs.

- **RRA:DEVPREDICT:period:index of DEVSEASONAL**
  - <period>: Es el número de puntos de datos primarios en el periodo de la temporada. Debe coincidir con el valor especificado por el argumento de <period>de HWPREDICT. Debe ser un valor entero mayor a 2.
  - <index of DEVSEASONAL>: Es el índice del arreglo DEVSEASONAL en el orden en el cual se especificarón los RRAs.
- **RRA:FAILURES:length:threshold:window length:rra-num**
  - <length>: Es el número de indicadores (valores de 0,1) que se almacenarán antes de cubrir la RRD. Un 1 indica una falla: esto es, el número de violaciones en las observaciones en la última ventana antes de cubrir o exceder el límite.
  - <threshold>: Es el número de violaciones máximas dentro de una ventana (valores observados fuera de los límites definidos) que constituyen una falla.
  - <window length>: Es el número de puntos en el tiempo de una ventana. Se especifica un número entero mayor o igual al límite, o menor o igual a 28 (el número máximo posible).
  - <index of DEVSEASONAL>: Es el índice del arreglo DEVSEASONAL en el orden en el cual se especificarón los RRAs.

Detectando y Gráficando el comportamiento aberrante

El algoritmo de detección de comportamiento aberrante no requiere más que el comando de actualización proporcionado por la herramieta de RRDTool. A continuación se define el código usado para detectar dicho comportamiento.

Actualización de los valores en la RRD

```
1  import time
2  import rrdtool
3  from getSNMP import consultaSNMP
4  from Notify import check_aberration
5  total_input_traffic = 0
6  total_output_traffic = 0
7
8  rrdpath="/home/tani/Escritorio/Trend-Non-Linear/"
9  pngpath="/home/tani/PycharmProjects/NoLineal/IMG/"
10 fname="netPred.rrd"
11 pngfname="predict.png"
12 title="Deteccion de comportamiento anomalo"
```

```
# Generate charts for last 24 hours
endDate = rrdtool.last(fname) #ultimo valor del XML
begDate = endDate - 86000

while 1:
    total_input_traffic = int(consultaSNMP('comunidadASR','localhost','1.3.
    total_output_traffic = int(consultaSNMP('comunidadASR','localhost','1.3

    valor = str(rrdtool.last(fname)+100)+":" + str(total_input_traffic) + '
    print valor
    ret = rrdtool.update(fname, valor)
    rrdtool.dump(fname,'netP.xml')
    time.sleep(1)
    print check_aberration(rrdpath,fname)

if ret:
    print rrdtool.error()
    time.sleep(300)
```

Gráficando el comportamiento

```
import time
import rrdtool

fname="netPred.rrd"
pngfname="predict.png"
title="Deteccion de comportamiento anomalo, valor de Alpha 0.1"
endDate = rrdtool.last(fname)
begDate = endDate - 86000

rrdtool.tune(fname,'--alpha','0.1')
ret = rrdtool.graph("netPalphaBajoFallas.png",
                    '--start', str(begDate), '--end', str(endDate), '--
                    "--vertical-label=Bytes/s",
                    '--slope-mode',
                    "DEF:obs="      + fname + ":inoctets:AVERAGE",
                    "DEF:outoctets=" + fname + ":outoctets:AVERAGE",
                    "DEF:pred="     + fname + ":inoctets:HWPREDICT",
                    "DEF:dev="      + fname + ":inoctets:DEVPREDICT",
                    "DEF:fail="     + fname + ":inoctets:FAILURES",
                    "CDEF:scaledobs=obs,8,*",
                    "CDEF:upper=pred,dev,2,*,+",
                    "CDEF:lower=pred,dev,2,*,-",
                    "CDEF:scaledupper=upper,8,*",
```

```
24                  "CDEF:scaledlower=lower,8,*",
25                  "CDEF:scaledpred=pred,8,*",
26                  "TICK:fail#FDD017:1.0:Fallas",
27                  "LINE3:scaledobs#00FF00:In traffic",
28                  "LINE1:scaledpred#FF00FF:Prediccion\\n",
29                  "LINE1:scaledupper#ff0000:Upper Bound Average bits
30                  "LINE1:scaledlower#0000FF:Lower Bound Average bits
```

Finalmente en la imagen 5.17 **Comportamiento en red con un alpha bajo** se muestra la detección del comportamiento anómalo en el tráfico de una red.

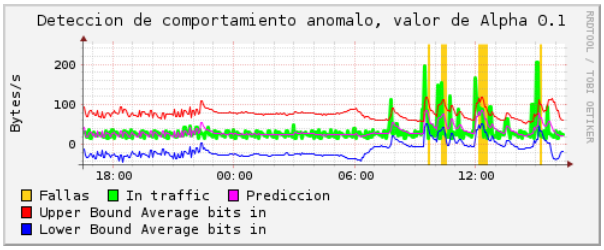


Figura 5.17: Detección de Comportamiento Anómalo.

6 | Detección y Notificación de Comportamiento Anómalo

6.1. Configuración del Agente

Para detectar y notificar el comportamiento anómalo de una variable de un cliente en específico, se debe proporcionar las especifícaciones de la variable y del agente en cuestión.

Recepción de Parámetros de Configuración

```
#! /usr/bin/python3
from getSNMP import *
from rrdUpdate import *
from rrdCreateDatabase import *
import threading

def startEvalDirect(agent_ip, port, comunity_name, oid, mode):
    if mode == "1":
        #Se seleciono un SNMPGET a una oid en especifico, primero verificamos si se puede leer
        consulta = consultaSNMP(comunity_name, agent_ip, port, oid)

        if (consulta == "None" or consulta == ""):
            return "NO SUCH INSTANCE CURRENTLY EXISTS AT THIS OID - FAILURE"
        else:
            database_name = raw_input("Inserte el nombre de la base de datos: ")

            if createDatabaseExam(database_name) == True:
                updateThread = threading.Thread(target = updateDirectGetDatabase,
                    name = "Update - Worker",
                    args = (database_name, comunity_name, agent_ip, port, oid))
                updateThread.start()

                return "SUCCESS"

            else:
                return "DATABASE CREATION - FAILURE"

    elif mode == "2":
        #Se seleciono un SNMPWALK a una oid en especifico, primero verificamos si se puede leer
        consulta = consultaWALKSNMP(comunity_name, agent_ip, port, oid)

        if (consulta == "None" or consulta == ""):
            return "NO SUCH INSTANCE CURRENTLY EXISTS AT THIS OID - FAILURE"

        else:
            database_name = raw_input("Inserte el nombre de la base de datos: ")

            if createDatabase(database_name, len(consulta)) == True:
                updateThread = threading.Thread(target = updateDirectWalkDatabase,
                    name = "Update - Worker",
                    args = (database_name, comunity_name, agent_ip, port, oid))
                updateThread.start()

                return "SUCCESS"

            else:
                return "DATABASE CREATION - FAILURE"
    else:
        return "BAD MODE - FAILURE"

def startEvalPercentage(agent_ip, port, comunity_name, oid_top, oid_variable, mode):
    if mode == "1":
        #Se seleciono un SNMPGET a una oid en especifico, primero verificamos si se puede leer
        consulta1 = consultaSNMP(comunity_name, agent_ip, port, oid_top)
```

```
55          consulta2 = consultaSNMP(comunity_name, agent_ip, port, oid_variable)
56
57          if (consulta1 == "None" or consulta1 == "" or consulta2 == "None" or consulta2 == ""):
58              return "NO SUCH INSTANCE CURRENTLY EXISTS AT THIS OID - FAILURE"
59          else:
60              database_name = raw_input("Inserte el nombre de la base de datos: ")
61
62              if createDatabase(database_name, 1) == True:
63                  updateThread = threading.Thread(target = updatePercentageGetDatabase,
64                      name = "Update - Worker",
65                      args = (database_name, comunity_name, agent_ip, port, oid_top, oid_variable))
66                  updateThread.start()
67
68              return "READ SUCESS"
69
70      elif mode == "2":
71          #Se seleciono un SNMPWALK a una oid en especifico, primero verificamos si se puede leer
72          consulta1 = consultaWALKSNMP(comunity_name, agent_ip, port, oid_top)
73          consulta2 = consultaWALKSNMP(comunity_name, agent_ip, port, oid_variable)
74
75          if (consulta1 == "None" or consulta1 == "" or consulta2 == "None" or consulta2 == "" or len(consulta1) != len(consulta2
76              return "NO SUCH INSTANCE CURRENTLY EXISTS AT THIS OID - FAILURE"
77
78          else:
79              for i in range(len(consulta1)):
80                  valor = "N:" + str((consulta2[i]*100)/consulta1[i])
81                  print valor
82              return "READ SUCESS"
83
84      else:
85          return "BAD MODE - FAILURE"
86  '''
87  def startEvalIO(agent_ip, port, comunity_name, oid_input, oid_output, mode):
88      if mode == "1":
89          #Se seleciono un SNMPGET a una oid en especifico, primero verificamos si se puede leer
90          consulta1 = consultaSNMP(comunity_name, agent_ip, port, oid_input)
91          consulta2 = consultaSNMP(comunity_name, agent_ip, port, oid_output)
92
93          if (consulta1 == "None" or consulta1 == "" or consulta2 == "None" or consulta2 == ""):
94              return "NO SUCH INSTANCE CURRENTLY EXISTS AT THIS OID - FAILURE"
95          else:
96              variable_input =  int(consultaSNMP(comunity_name, agent_ip, port, oid_top))
97              variable_output = int(consultaSNMP(comunity_name, agent_ip, port, oid_variable))
98              valor = "N:" + str(variable_input) + ":" + str(variable_output)
99              print(valor)
100             return "READ SUCCESS"
101
102     elif mode == "2":
103         #Se seleciono un SNMPWALK a una oid en especifico, primero verificamos si se puede leer
104         consulta1 = consultaWALKSNMP(comunity_name, agent_ip, port, oid_input)
105         consulta2 = consultaWALKSNMP(comunity_name, agent_ip, port, oid_output)
106
107         if (consulta1 == "None" or consulta1 == "" or consulta2 == "None" or consulta2 == "" or len(consulta1) != len(consulta2
108             return "NO SUCH INSTANCE CURRENTLY EXISTS AT THIS OID - FAILURE"
109
110         else:
111             for i in range(len(consulta1)):
112                 valor = "N:" + str(consulta1[i] + ":" + consulta2[i])
113                 print(valor)
114             return "READ SUCCESS"
115
116     else:
117         return "BAD MODE - FAILURE"
118 '''
119
120
121 def main():
122     agent_ip = raw_input("Inserte la IP del Agente: ")
123     port = raw_input("Inserte el puerto: ")
124     comunity_name = raw_input("Inserte el nombre de la comunidad del agente: ")
125
126     print "\nInserte el tipo de valor que desea obtener: \n 1) Directo \n 2) Porcentaje \n 3) I/O"
127     type_val = raw_input()
128
129     if type_val == "1":
130         oid = raw_input("Inserte el OID de la variable que desea leer: ")
131         print "\nInserte el modo en el cual se realizara la consulta SNMP: \n 1) GET \n 2) WALK"
132         mode = raw_input()
133         print startEvalDirect(agent_ip, port, comunity_name, oid, mode)
134
135     elif type_val == "2":
136         oid_top = raw_input("Inserte el OID del valor tope que desea leer: ")
137         oid_variable = raw_input("Inserte el OID del valor variable que desea leer: ")
138         print "\nInserte el modo en el cual se realizara la consulta SNMP: \n 1) GET \n 2) WALK"
139         mode = raw_input()
140         print startEvalPercentage(agent_ip, port, comunity_name, oid_top, oid_variable, mode)
141
142     elif type_val == "3":
143         oid_input = raw_input("Inserte el OID de la variable de entrada que desea leer: ")
144         oid_output = raw_input("Inserte el OID de la variable de salida que desea leer: ")
145         print("\nInserte el modo en el cual se realizara la consulta SNMP: \n 1) GET \n 2) WALK")
```

```
146             mode = raw_input()
147             print startEvalIO(agent_ip, port, comunity_name, oid_input, oid_output, mode)
148         else:
149             print "BAD TYPE - FAILURE"
150 
151 if __name__ == "__main__":
152     main()
```

En la imagen 6.1 **Configuración en Consola** se muestra como se introducen los parámetros de configuración del Agente:

Figura 6.1: Configuración en Consola.

6.2. Detección de Comportamiento Anómalo

6.2.1. Creación de la RRD

Una vez configurado el agente y que haya exitosamente realizado una consulta SNMP, se procederá a crear la RRD, se deberá proporcionar el tipo y nombre de la variable.

```
1  import rrdtool
2  
3  def createDatabase(database_name, num_sources):
4      try:
5          datasources = []
6          rraverages = []
7  
8          for i in range(num_sources):
9              data_string = "DS:VALUES" + str(i+1) + ":COUNTER:600:U:U"
10             datasources.append(data_string)
11             rraverages.append("RRA:AVERAGE:0.5:1:1000")
12 
13         print "datasources -> " + str(datasources)
14         print "rraverages -> " + str(rraverages)
15 
16         ret = rrdtool.create("RRD/" + database_name + str(".rrd"),
17                              "--start",'N',
18                              "--step",'1',
19                              datasources,
20                              rraverages,
21                              #RRA:HWPREDICT:rows:alpha:beta:seasonal period[:rra-num]
22                              "RRA:HWPREDICT:30000:0.1:0.0035:60:" + str(num_sources + 2),
23                              #RRA:SEASONAL:seasonal period:gamma:rra-num
24                              "RRA:SEASONAL:60:0.1:" + str(num_sources + 1),
```

```
25                                  #RRA:DEVSEASONAL:seasonal period:gamma:rra-num
26                                  "RRA:DEVSEASONAL:60:0.1:" + str(num_sources + 1),
27                                  #RRA:DEVPREDICT:rows:rra-num
28                                  "RRA:DEVPREDICT:30000:" + str(num_sources + 3),
29                                  #RRA:FAILURES:rows:threshold:window length:rra-num
30                                  "RRA:FAILURES:30000:7:9:" + str(num_sources + 3))
31
32          if ret:
33              print rrdtool.error()
34          return True
35
36      except ValueError:
37          return False
38
39  def createDatabaseExam(database_name):
40      try:
41          ret = rrdtool.create("RRD/" + database_name + str(".rrd"),
42                                  "--start",'N',
43                                  "--step",'1',
44                                  "DS:VALUES1:COUNTER:600:U:U",
45                                  "RRA:AVERAGE:0.5:1:1000",
46                                  #RRA:HWPREDICT:rows:alpha:beta:seasonal period[:rra - num]
47                                  "RRA:HWPREDICT:30000:0.1:0.0035:60:3",
48                                  #RRA:SEASONAL:seasonal period:gamma:rra-num
49                                  "RRA:SEASONAL:60:0.1:2",
50                                  #RRA:DEVSEASONAL:seasonal period:gamma:rra-num
51                                  "RRA:DEVSEASONAL:60:0.1:2",
52                                  #RRA:DEVPREDICT:rows:rra-num
53                                  "RRA:DEVPREDICT:30000:4",
54                                  #RRA:FAILURES:rows:threshold:window length:rra-num
55                                  "RRA:FAILURES:30000:7:9:4")
56
57          if ret:
58              print rrdtool.error()
59          return True
60
61      except ValueError:
62          return False
```

Al crearse la RRD se ejecutan los workers para la lectura y actualización de los valores de la variable.

6.2.2. Lectura y Actualización de Valores

```
import time
import logging
import rrdtool

from getSNMP import *
from notification import *

logging.basicConfig( level=logging.DEBUG,
    format='[%(levelname)s] - %(threadName)-10s : %(message)s')

counterfail = 0
now = 0
activated = 0
future = now + 10

def graph(database_name, num_graphs, labels_name, init_time):
    for i in range(num_graphs):
        ret = rrdtool.graph("IMG/" + database_name + ".png",
                            "--title=ABERRANT BEHAVIOUR FOR " + labels_name[1].upper() + " " + str(i+1),
                            "--start", str(init_time),
                            "--end", str(rrdtool.last("RRD/" + database_name + '.rrd')),
                            "--vertical-label=" + labels_name[0],
                            "--width=1000",
                            "--height=500",
                            "DEF:obs=" + "RRD/" + database_name + ".rrd:VALUES" + str(i+1) + ":AVERAGE",
                            "DEF:pred=" + "RRD/" + database_name + ".rrd:VALUES" + str(i+1) + ":HWPREDICT",
                            "DEF:dev=" + "RRD/" + database_name + ".rrd:VALUES" + str(i+1) + ":DEVPREDICT",
                            "DEF:fail=" + "RRD/" + database_name + ".rrd:VALUES" + str(i+1) + ":FAILURES",
                            "CDEF:pfail=fail,1,*",
                            "CDEF:scaledobs=obs,8,*",
                            "CDEF:upper=pred,dev,2,*,+",
                            "CDEF:lower=pred,dev,2,*,-",
                            "CDEF:scaledupper=upper,8,*",
                            "CDEF:scaledlower=lower,8,*",
                            "TICK:fail#FDD017:1.0:Failures",
                            "CDEF:scaledpred=pred,8,*",
                            "LINE1:scaledobs#00FF00:" + labels_name[1],
                            "LINE2:scaledpred#ee0099:Forecast",
                            "LINE1:scaledupper#FF000E:Upper Bound",
                            "LINE1:scaledlower#0012FF:Lower Bound",
                            "PRINT:pfail:LAST:%0.0lf")
                           #"LINE1:obs#00FF00:" + labels_name[1],
                           #"LINE2:pred#ee0099:Forecast",
                           #"LINE1:upper#FF000E:Upper Bound",
                            #"LINE1:lower#0012FF:Lower Bound")

        logging.debug(ret)

        global now
        global future
        global activated

        now = time.time()
        to_normalformat = time.ctime(int(now))

        if(ret[2][0] != 'nan'):
                failure = int(ret[2][0])
                if(failure and (now > future)):
                    logging.debug("There is an error")
                    future = now + 30
                    activated = 1
                    sendEmail("Aberrant Behaviour Detetcted!! at: " + str(to_normalformat), database_name + ".png")

                elif (failure == 0 and activated == 1):
                    logging.debug("The error has finished")
                    activated = 0
                    sendEmail("Error finished at: " + str(to_normalformat), database_name + ".png")

                logging.debug(to_normalformat)

def updateDirectGetDatabase(database_name, comunity_name, agent_ip, port, oid):
    variable_read = 0
    init_time = rrdtool.last("RRD/" + database_name + str(".rrd"))

    labels = raw_input("Inserte el tipo de variable y el nombre: ").split(',')
    print labels

    if len(labels) == 2:
        while 1:
            variable_read = int(consultaSNMP(comunity_name, agent_ip, port, oid))
            value = "N:" + str(variable_read)
            logging.debug(value)
            ret = rrdtool.update("RRD/" + database_name + str(".rrd"), value)
            rrdtool.dump("RRD/" + database_name + str(".rrd"), "XML/" + database_name + str(".xml"))

            graph(database_name, 1, labels, init_time)
            time.sleep(1)
```

```
88
89
90  def updateDirectWalkDatabase(database_name, comunity_name, agent_ip, port, oid):
91      variable_read = []
92      init_time = rrdtool.last("RRD/" + database_name + str(".rrd"))
93
94      labels = raw_input("Inserte el tipo de variable y el nombre: ").split(',')
95      print labels
96
97      if len(labels) == 2:
98          while 1:
99              variable_read = consultaWALKSNMP(comunity_name, agent_ip, port, oid)
100             value = "N"
101
102             for variable in variable_read:
103                 value = value + ":" + str(variable)
104
105             logging.debug(value)
106             ret = rrdtool.update("RRD/" + database_name + str(".rrd"), value)
107             rrdtool.dump("RRD/" + database_name + str(".rrd"), "XML/" + database_name + str(".xml"))
108
109             time.sleep(1)
110             graph(database_name, len(variable_read), labels, init_time)
111
112 def updatePercentageGetDatabase(database_name, comunity_name, agent_ip, port, oid_top, oid_variable):
113     variable_read = 0
114     init_time = rrdtool.last("RRD/" + database_name + str(".rrd"))
115
116     variable_top =  int(consultaSNMP(comunity_name, agent_ip, port, oid_top))
117
118     labels = raw_input("Inserte el tipo de variable y el nombre: ").split(',')
119     print labels
120
121     if len(labels) == 2:
122         while 1:
123             variable_read = int(consultaSNMP(comunity_name, agent_ip, port, oid_variable))
124             value = "N:" + str(int((variable_read*100)/variable_top))
125
126             logging.debug(value)
127             ret = rrdtool.update("RRD/" + database_name + str(".rrd"), value)
128             rrdtool.dump("RRD/" + database_name + str(".rrd"), "XML/" + database_name + str(".xml"))
129
130             time.sleep(1)
131             graph(database_name, 1, labels, init_time)
```

En la imagen 6.2 **Lectura y Actualización** se muestra como se ejecutan los workers que realizan la lectura y actualización de los valores de la variable monitoreada:

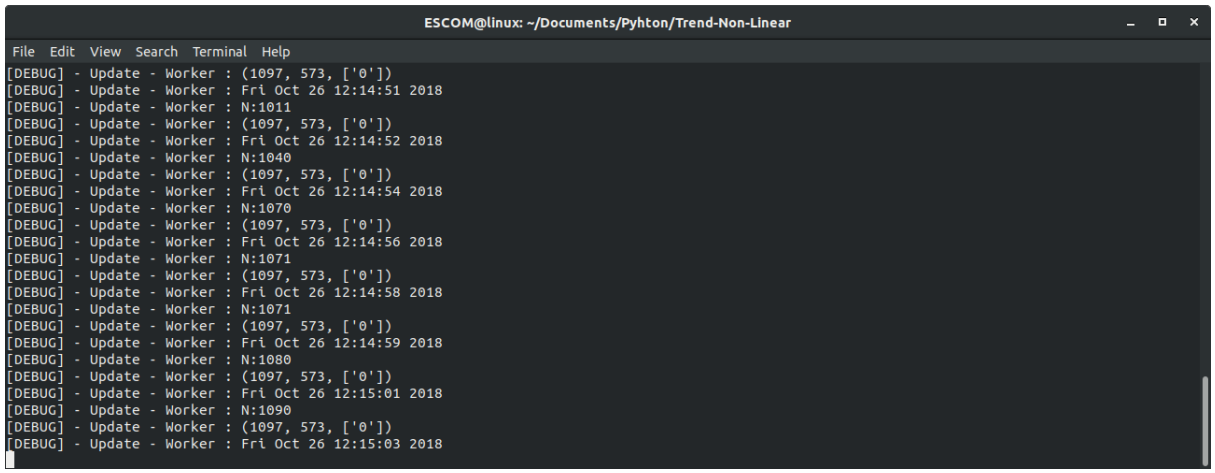


Figura 6.2: Lectura y Actualización.

6.3. Notificación de Error

```
import smtplib
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart
import time


pngpath = '/home/ESCOM/Documents/HW/IMG/' #'/home/osboxes/Desktop/TrendLineal/'

#--------------------Envio de correos
def sendEmail(subject, imag):

    msg = MIMEMultipart()

    msg['Subject'] = subject


    msg['From'] = 'escomcec@gmail.com'
    msg['To'] = 'tanibet.escom@gmail.com'
    password = "cec12345"

    fp = open(pngpath+imag, 'rb')

    msg.attach(MIMEImage(fp.read()))


    s = smtplib.SMTP('smtp.gmail.com: 587')
    s.starttls()

    # Login Credentials for sending the mail
    s.login(msg['From'], password)


    # send the message via the server.
    s.sendmail(msg['From'], msg['To'], msg.as_string())

    s.quit()

    print "successfully sent email to %s:" %(msg['To'])
```

Por último cuando se detecta una falla en la red, se le notifica al administrador vía email que ha ocurrido un error; el servicio envía una imagen adjunta mostrando el comportamiento de la red y la falla detectada. También se le notifica al administrador cuando la falla ha terminado y la duración de la misma.

A continuación se muestrán las imagenes mostrando el **monitoreo**, **detección**, **notificación** y **finalización** de una falla.

En la imagen 6.3 **Monitoreo del Agente** se muestra el monitoreo de la variable del agente.

Figura 6.3: Monitoreo del Agente.

En la imagen 6.4 **Detección de Falla** se muestra la detección de una falla.

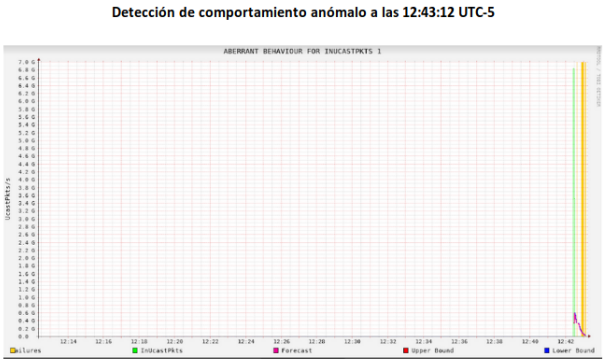


Figura 6.4: Detección de Falla.

En la imagen 6.5 **Notificación de Falla** se muestra como se le notifica al administrador de red que ha ocurrido una falla.

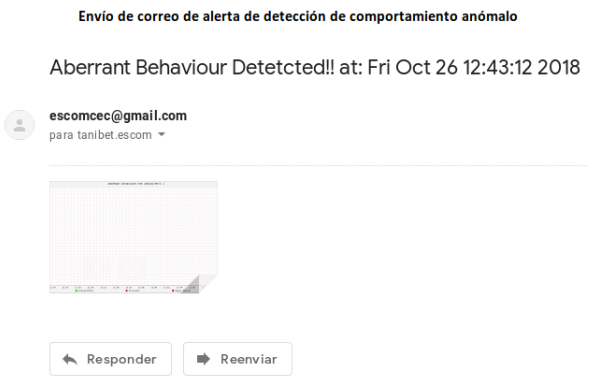


Figura 6.5: Notificación de Falla.

En la imagen 6.6 **Fin de Falla** se muestra cuando una falla ha finalizado.

Figura 6.6: Fin de Falla.

En la imagen 6.7 **Notificación de Fin de Falla** se muestra cuando una falla ha finalizado.

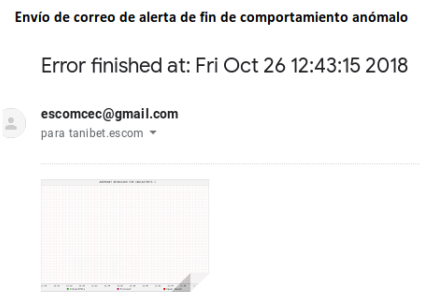


Figura 6.7: Notifiación de Fin de Falla.

7 | Conclusiones

7.1. Amador Nava Miguel Ángel

En la gestión de servicios en red, es una herramienta muy útil el Protocolo simple de administración de red (SNMP) para la obtención de información de los distintos equipos que pertenecen a la red como pudimos verlo en el desarrollo de la práctica, independientemente del lenguaje de programación en el cual se desee implementar una herramienta, los métodos matematicos para predicciones lineales y no líneales así como la notificación en el tiempo adecuado es de vital importancia para asegurar el trabajo eficiente de la red. A manera didactica se implementó en python si se deseara mayor velocidad en el manejo de la información podriamos optar por otro lenguaje de programación como lo es C++.

7.2. López Ayala Eric Alejandro

Dentro del área de la administración de los servicios en red es necesario tener la capacidad de administrar fallas que se pueden presentar en la red. Esta práctica hizo evidente que es necesario a la hora de proporcionar un servicio de administración en red, el establecer las variables y el entorno en los cuales se proporcionará el servicio, mediante la definición de un contrato de nivel de servicio.

Para la detección de estas fallas se implementarón distintos mecanismos, los cuales al desarrollarlos se hizo evidente las desventajas y ventajas que cada uno nos ofrece. Esto me fue útil, ya que me proporcionó los conocimientos para poder decidir cuando hacer uso de cada método y decidir sobre cual variable aplicarlos. A su vez la notificación y manejo de estas fallas constituyen elementos muy importantes para todo administrador de red.

Por último la implementación de métodos matemáticos como lo fue mínimos cuadrados y Holt-Winters para la predicción, hizo que el análisis de las variables monitoreadas fuera más preciso y certero. En lo personal fue interesante la implementación del análisis de series en el tiempo en problemas prácticos.

7.3. López Romero Joel

De acuerdo a lo realizado en esta práctica, para la administración y monitorización de agentes dentro de una red, el empleo de métodos para analizar el comportamiento de ciertos dispositivos

dentro de un agente e inclusive para predecir este comportamiento, es importante, debido a las fallas que pueden ocurrir en un agente dentro de una red. Por ello, en esta práctica se presenta el desarrollo e implementación de tres métodos para analizar, prevenir y detectar errores en ciertos dispositivos de un agente dentro de una red usando un protocolo en común SNMP, además de la MIB.

El primer método *Línea Base*, en este método definimos 3 umbrales para monitorizar el comportamiento de 4 dispositivos del agente definido en el contrato, el cual se trata de una máquina virtual. El primer umbral lo definimos de al comportamiento analizado durante un periodo de tiempo obteniendo su desviación y promedio, a este promedio le sumamos un 5 % para evitar que siempre se esté en este umbral. El tercer umbral lo define el fabricante del dispositivo. Y el segundo umbral es el valor medio entre el primer y tercer umbral. Además de esto, se notifica cuando el ultimo valor supera algun umbral, la notificación es un correo al administrador.

El segundo método es *mínimos cuadrados*. Este método consta en dar una pendiente de acuerdo al comportamiento de un objeto de la MIB de un agente usando este método, está pendiente nos sirve para determinar o proyectar la fecha de la intersección con algún umbral dado.

El ultimo método, *Holt Winters* este método consta en un suavizamiento exponencial para pronosticar puntos de datos de una seria, para realizar esto se debe proporcionar las especificaciones de las variables Beta, Gama y alfa de una agente. Por medio de este método detectamos aberraciones de algún objeto de la MIB de un agente monitorizado, al igual que línea base, cuando existe tal aberración se notifica al administrador, notificando el inicio de la aberración y el fin.

Esta práctica tiene un nivel de complejidad mayor que la práctica anterior, tuvimos varios problemas en cuanto a la implementación, pero con perseverancia, investigación y tiempo, los fuimos resolviendo hasta lograr el objetivo.

8 | Glosario

Series: Una serie es simplemente una secuencia ordenada de números, cuyas únicas propiedades de cada punto de la serie son su *valor* y su *orden*.

Es útil pensar en una serie como una lista bidimensional de coordenadas *(x,y)*, donde x es el orden, y y es el valor.

Error: Un error es la diferencia entre lo *observado* y lo *esperado*, es un indicador indispensable de la exactitud de un método.

SSE: SSE (Sum Squared Errors, por sus siglás en inglés), es la suma de los errores cuadráticos entre los puntos de una serie.

MSE: MSE (Mean Squared Error, por sus siglás en inglés), es el error cuadrático medio, el cual es simplemente la raíz del SSE.

Level: Es un sinónimo para referirnos al valor esperado (con lo que respecta a este documento), referido con el simbolo l.

Trend: Trend es la pendiente formada por 2 puntos, definida por la fórmula:

$$m = \frac{\delta y}{\delta x} \tag{8.1}$$

Donde la delta y es la diferencia entre las coordenadas en y y delta x es la diferencia entre las coordenadas en x, para las series de tiempo la diferencia simpre es de 1, por lo que la ecuación se simplifica de la siguiente forma:

$$b = y_{\text{x}} - y_{\text{x-1}} \tag{8.2}$$

Temporada: Si una serie aparenta ser repetitiva en intervalos regulares, nos referimos a dicho intervalo como *temporada*. La temporalidad es requerida para que el método Holt-Winters funcione (ya que series no periodicas no pueden ser predecidas usando este método).

Longitud de temporada: Es el número de puntos de datos después de los cuales una nueva temporada comienza, se utiliza la letra L para denotar la longitud de la temporada.

Componente de temporada: Es la desviación adicional del nivel más la tendencia y que se

repite así mismo en el mismo offset dentro de una temporada.

Bibliografía

[1] P. Grillo. Rfc 1514. [Online]. Available: https://tools.ietf.org/html/rfc1514#section-3

[2] J. Brutlag. Notes on rrdtool implementation of aberrant behavior detection. [Online]. Available: http://cricket.sourceforge.net/aberrant/rrd_hw.htm

[3] G. Trubetskoy. Holt-winters forecasting for dummies - part i. [Online]. Available: https://grisha.org/blog/2016/02/16/triple-exponential-smoothing-forecasting-part-i/

[4] ——. Holt-winters forecasting for dummies - part ii. [Online]. Available: https://grisha.org/blog/2016/02/16/triple-exponential-smoothing-forecasting-part-ii/

[5] ——. Holt-winters forecasting for dummies - part iii. [Online]. Available: https://grisha.org/blog/2016/02/16/triple-exponential-smoothing-forecasting-part-iii/