Artículo
Kurro Lopez · Abr 13, 2021 Lectura de 7 min

Visualizando la jungla de datos - Parte II. Más fuentes y una mejor salida

En el artículo anterior creamos un gráfico simple con los datos de un solo archivo. Ahora bien, como todos sabemos, a veces tenemos diferentes archivos de datos para analizar y correlacionar. Así que en este artículo vamos a cargar datos adicionales de perfmon y aprenderemos a representarlos en el mismo gráfico.

Como podemos querer utilizar en informes o en una página web nuestros gráficos generados, también revisaremos formas de exportar los gráficos generados.

Cargando datos de Windows Perfmon

Los datos de perfmon extraídos del informe estándar de pbuttons tienen un formato de datos un poco peculiar. A primera vista es un archivo csv bastante sencillo. La primera fila contiene los encabezados de las columnas, las filas siguientes los datapoints. Sin embargo, para nuestros propósitos tendremos que hacer algo con las comillas que rodean las entradas de los valores. Utilizando el enfoque estándar para analizar el archivo en python, terminaremos con columnas de objetos de tipo cadena, que no funcionan bien para ser representados de forma gráfica.

```
perfmonfile="../vis-part2/perfmon.txt"
perfmon=pd.read_csv(perfmonfile,
                    header=0,
                    index_col=0,
                    converters={0: parse_datetime
                    })
```

A diferencia del archivo mgstat que utilizamos en la primera parte, los nombres de los encabezados están en la primera fila. Queremos que la primera columna defina nuestro índice (de esta manera no tenemos que volver a indexar el dataframe como hicimos la última vez). Por último, debemos analizar la primera columna para que realmente represente un DateTime. De lo contrario, terminaríamos con un índice de cadenas. Para ello, definimos una pequeña función de ayuda para analizar las fechas de perfmon:

```
def parse_datetime(x):
    dt = datetime.strptime(x, '%m/%d/%Y %H:%M:%S.%f')

    return dt
```

El parámetro ~~~conventers~~~ nos permite pasarlo como controlador de la primera columna.

Después de ejecutar esto, terminamos con los datos de perfmon en un DataFrame:

```
<class 'pandas.core.frame.DataFrame'>
Index: 2104 entries, 01/03/2017 00:01:19.781 to 01/03/2017 17:32:51.957
```

```
Columns: 105 entries, \\WEBSERVER\Memory\Available MBytes to \\WEBSERVER\System\Proce
ssor Queue Length
dtypes: float64(1), int64(11), object(93)
memory usage: 1.7+ MB
```

Ten en cuenta que la mayoría de las columnas actualmente son un *object*. Para convertir estas columnas a un formato utilizable, emplearemos la función to_numeric. Aunque podríamos usar apply para llamarlo en cada columna, eso estropearía nuestro índice de nuevo. Así que vamos a trazar los datos directamente mientras se canaliza a través de eso.

Plotting

En esta ocasión, nos interesa representar el tiempo total privilegiado de todas las CPUs. Lamentablemente, los números de las columnas no son constantes y varían con el número de CPU y drives. Así que tendrás que observar y averiguar qué columna es. En mi ejemplo es 91:

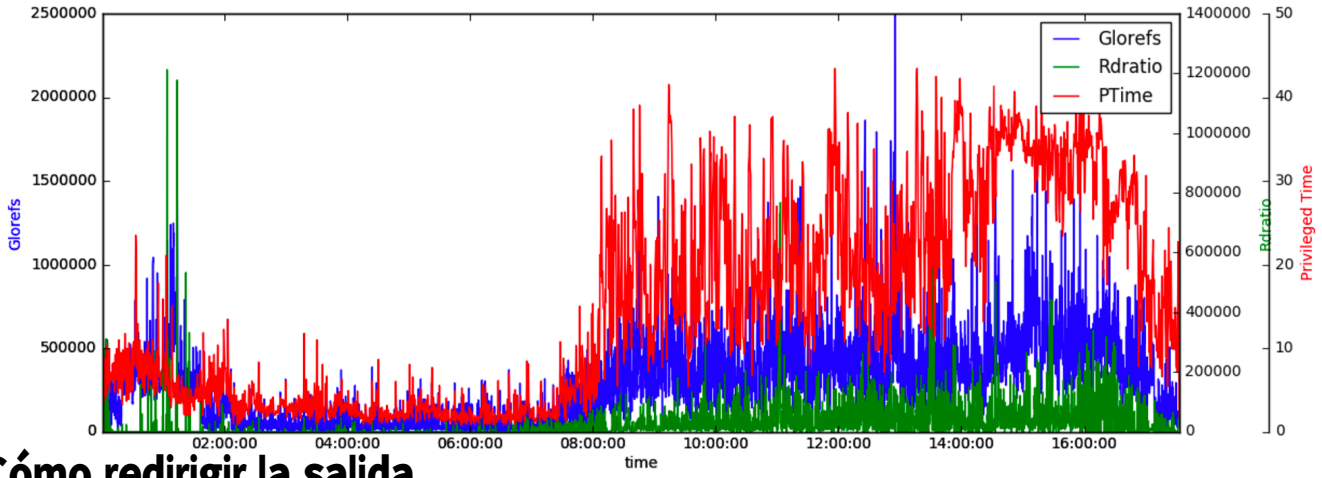
```
perfmon.columns[91]

'\\\\WEBSERVER\\Processor(_Total)\\% Privileged Time'
```

Utilizaremos más o menos el mismo enfoque que la última vez para crear un gráfico con Glorefs, Rdratio y nuestro nuevo Privileged Time:

```
plt.figure(num=None, figsize=(16,5), dpi=80, facecolor='w', edgecolor='k')
host = host_subplot(111, axes_class=AA.Axes)
plt.subplots_adjust(right=0.75)

par1 = host.twinx()
par2 = host.twinx()
offset = 60
new_fixed_axis = par2.get_grid_helper().new_fixed_axis
par2.axis["right"] = new_fixed_axis(loc="right",axes=par2,offset=(offset, 0))
par2.axis["right"].toggle(all=True)

host.set_xlabel("time")
host.set_ylabel("Glorefs")

par1.set_ylabel("Rdratio")
par2.set_ylabel("Privileged Time")
ws=30
p1,=host.plot(data.Glorefs,label="Glorefs")
p2,=par1.plot(data.Rdratio,label="Rdratio")
p3,=par2.plot(pd.to_numeric(perfmon[perfmon.columns[91]],errors='coerce'),label="PTim
e")

host.legend()

host.axis["left"].label.set_color(p1.get_color())
par1.axis["right"].label.set_color(p2.get_color())
par2.axis["right"].label.set_color(p3.get_color())

plt.draw()
plt.show()
```

Aquí es donde se utiliza *to_numeric*:

```
p3,=par2.plot(pd.to_numeric(perfmon[perfmon.columns[91]],errors='coerce'),label="PTim
e")
```



Cómo redirigir la salida

Mientras que Notebook es realmente bueno para tener un vistazo rápido de nuestros datos, con el tiempo nos gustaría ser capaces de ejecutar nuestros scripts de forma no interactiva, por lo que queremos que nuestros gráficos salgan como imágenes. Obviamente, realizar una captura de pantalla implica demasiado trabajo manual, así que utilizaremos la función pyplot *savefig()*.

Reemplazaremos las llamadas *draw()* y *show()* con la llamada *savefig()*:

```
#plt.draw()
#plt.show()
plt.savefig("ptime-out.png")
```

que nos dará el png en nuestro directorio de trabajo actual.

Salida avanzada

Como un pequeño ejercicio adicional vamos a echar un vistazo a Bokeh. Una de las muchas útiles características que bokeh está añadiendo a nuestra caja de herramientas, es la capacidad de generar nuestros gráficos como un archivo HTML interactivo. Interactivo en este caso significa que podemos ampliar y desplazarnos por nuestros datos. Añade la capacidad de vincular gráficos entre sí y podrás crear fácilmente renderizados interactivos de datos de pbuttons (u otros). Estos son especialmente útiles, porque se ejecutan en cualquier navegador y se pueden distribuir fácilmente a varias personas.

Por ahora, nuestro objetivo es añadir solo dos gráficos a nuestra salida. Nos gustaría obtener *Glorefs* y el tiempo privilegiado de perfom en una página.

Para eso primero tendremos que importar bokeh:

```
from bokeh.plotting import *
```

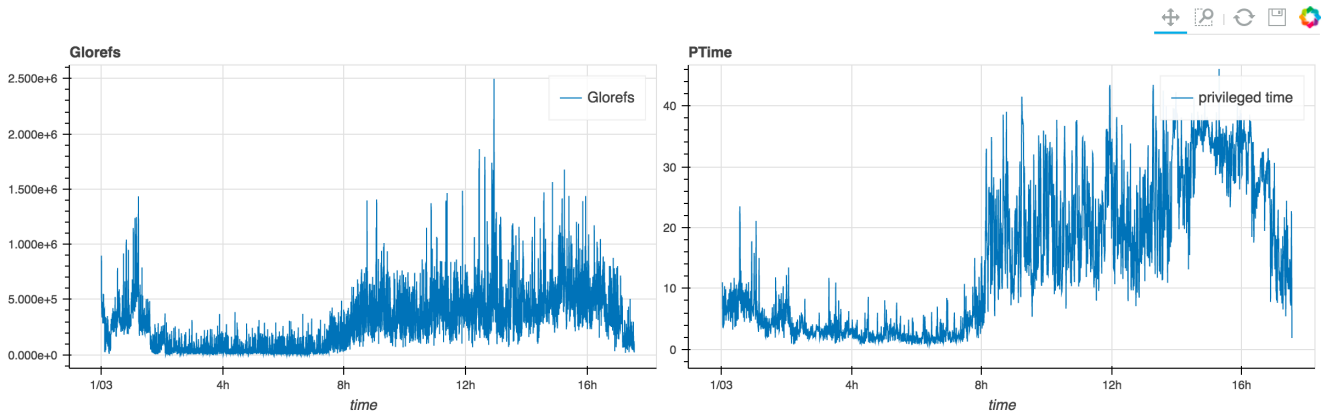
Vamos a definir un par de propiedades y etiquetas, así como el tamaño de cada gráfico. Después, representaremos los datos de los objetos que recogimos antes y ya hemos terminado.

Fíjate en el comentario *output_notebook()*, esto renderizaría la salida directamente en nuestro jupyter notebook. Estamos utilizando *output_file* ya que nos gustaría tener un archivo que podamos distribuir.

```
output_file("mgstat.html")
#output_notebook()
TOOLS="pan,box_zoom,reset,save"

left = figure(tools=TOOLS,x_axis_type='datetime',
    title="Glorefs",width=600, height=350,
   x_axis_label='time'
)
right=figure(tools=TOOLS,x_axis_type='datetime',
    title="PTime",width=600, height=350,
   x_axis_label='time',x_range=left.x_range
)
left.line(data.index,data.Glorefs,legend="Glorefs",line_width=1)
right.line(perfmon.index,pd.to_numeric(perfmon[perfmon.columns[91]],errors='coerce'),
legend="privileged time",line_width=1)
p=gridplot([[left,right]])
show(p)
```

El ingrediente clave aquí es la vinculación de los rangos de nuestros dos gráficos con *x_range=left.x_range*. Esto actualizará la ventana derecha con nuestra selección/zoom/movimiento desde la izquierda (y viceversa).

La lista de HERRAMIENTAS es solo la lista de herramientas que nos gustaría tener en nuestra pantalla resultante. Usando *gridplot* vamos a poner los dos gráficos uno al lado del otro:



También puedes echar un vistazo al html resultante en el repositorio de github. Parece que es demasiado grande para publicarlo directamente por medio de github, así que tendrás que descargarlo.

Conclusiones

En este artículo, exploramos la extracción de datos de diferentes fuentes y su representación en el mismo gráfico. También estamos gestionando datos con frecuencia de muestreo diferente (apuesto a que no te diste cuenta ;D). Bokeh nos ofrece una potente herramienta para crear vistas interactivas fácilmente distribuibles para nuestros gráficos.

En próximos artículos exploraremos más cosas para representar gráficamente: csp.log, registros de acceso de

apache/iis, eventos de cconsole.log. Si tienes alguna sugerencia sobre algunos datos que te gustaría ver procesados con python, no dudes en comentarlo.

¡ Comparte tus experiencias! ¡ Esto está pensado como un aprendizaje interactivo!

Enlaces

Puedes encontrar todos los archivos de este artículo aquí
También puedes ver la nueva herramienta de extracción pButtons de @murrayo, basada en algunas de las técnicas comentadas: https://github.com/murrayo/yape

#Herramientas #Python #Rendimiento #Visualización #Caché

URL de fuente: https://es.community.intersystems.com/post/visualizando-la-jungla-de-datos-parte-ii-m%C3%A1s-fuentes-y-una-mejor-salida