記事
Toshihiko Minamoto · 2021年12月21日 7m read

データジャングルの視覚化 -- パート2
より良くのソースでより優れた出力を！

先週 のディスカッションでは、1つのファイルのデータ入力に基く単純なグラフを作成しました。
ご存知のように、解析して相関付けるデータファイルが複数あることがあります。そこで今週は、perfmonデータを追加して読み込み、それを同じグラフにプロットする方法について学習しましょう。生成したグラフをレポートやWebページで使用する可能性あるため、生成したグラフのエクスポート方法についても説明します。

WindowsのperfmonデータをPythonに読み込む

標準のpButtonsレポートから抽出されたperfmonデータは、少し独特なデータ形式です。
一見すると、かなり単純なCSVファイルで、
最初の行には列のヘッダーがあり、それ以降の行にはデータポイントが含まれています。
ただし、ここでの目的のために、値エントリーを囲む引用符をどうにかする必要があります。標準的なアプローチを使用してファイルをPythonに解析すると、文字列オブジェクトの列ができてしまい、うまくグラフ化できません。

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

パート1で使用したmgstatファイルは異なり、ヘッダー名は最初の行にあります。最初の列にインデックスを定義しようと思います（こうすれば、前回のようにデータフレームのインデックスを再作成なくて済むためです）。最後に、実際にDateTimeを表すように最初の列を解析する必要があります。
そうでない場合、文字列インデックスになってしまうためです。
これを行うために、perfmonの日付を解析するヘルパー関数を定義します。

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

    return dt
```

~~~converters~~~パラメータを使用すると、最初の列のハンドラとして渡すことができます。

これを実行したら、DataFrameでperfmonデータが得られました。

```
<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
```

大半の列は現在　オブジェクト　であることに 注 意してください。
これらの列を使用可能な形式に変換するために、 to_numeric関数を使用します。 apply
を使用して各列にこの関数を呼び出すことも可能ですが、それではインデックスがまた台無しになってしまいます
。そこで、データにその関数を適用しながら直接データをプロットすることにします。

プロット

この実行では、全CPUの合計特権時間をプロットすることに関心があります。
残念ながら、列番号は一定でなく、CPUとドライブの数によって異なります。
そのため、それがどの列かを調べて知る必要があります。私の例では、91です。

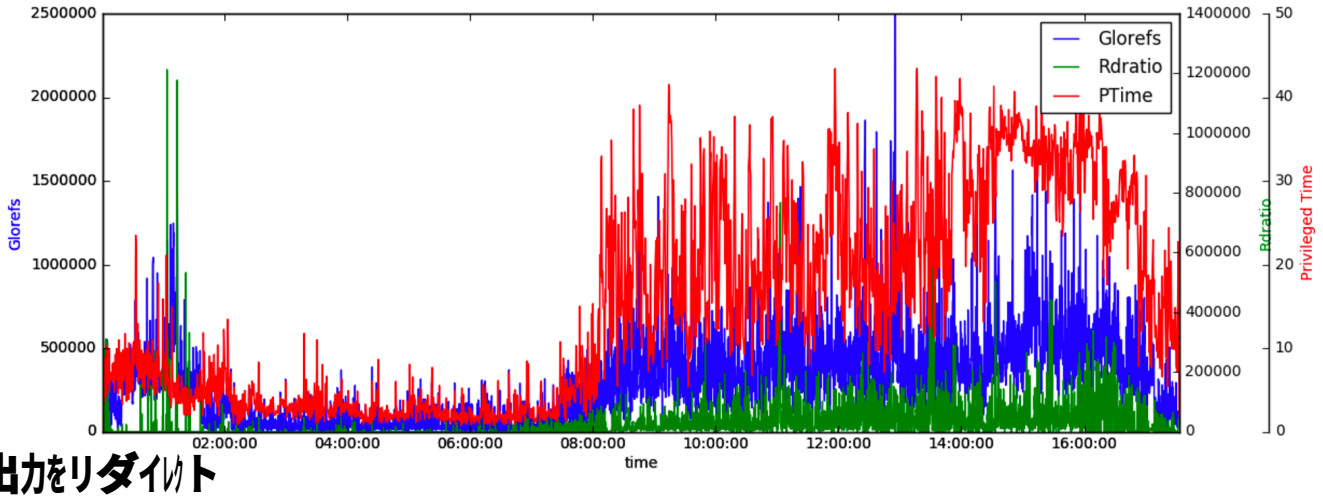
```
perfmon.columns[91]

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

前回同じアプローチを使用して、Glorefs、Rdratio、および新しいPriviledged Timeでグラフを作成ます。

```
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()
```

ここで *to_numeric*を使用することができます。

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



出力をリダイレクト

このノートブックは、データを一目で確認するのに非常に優れていますが、最終は、スクリプトを非対話的に実行できるようにしたいため、グラフをイメージとして出力したいと思います。
スクリーンショットには、明らかに伴う手作業が多くなりすぎるため、puplot関数の *savefig()*を使用します。

*draw()*と *show()*の呼び出しを *savefig()*呼び出しに置き換えます。

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

こうすると、現在の作業ディレクトリにpngが表示されます。

高度な出力

ちょっとした追加演習として、 Bokehを見てみましょう。Bokehがツールボックスに追加する多数の優れた機能の1つに、グラフをインタラクティブなHTMLファイルとして出力する機能があります。
インタラクティブとは、この場合、データをスクロールしてズームインできることを指します。グラフ同士を関連付ける機能を追加すると、pButtonsデータ（またはその他のデータ）のインタラクティブなレンダリングを簡単に作成することができます。
これらは、あらゆる最新のブラウザで実行し、多数の人に簡単に配布できるため、特に便利です。

今のところは、2 つのグラフを出力に追加することを狙いとしています。
perfmonの *Glorefs*と特権時間を1つのページにまとめたいと思います。

そのためにはまず、Bokehをインポートする必要があります。

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

いくつかのプロパティとラベルを定義し、各プロットのサイズも決定します。
その後で、先に収集していたデータオブジェクトをプロットすれば、それで完了です。

コメントアウトされている *output_notebook()*
は、Jupyterノートブックに直接出力をレンダリングすることに 注 意してください。
最終ファイルを配布できるようにするため、 *output_file*を使用しています。

```
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)
```

ここで重要なのは、2つのグラフの範囲の関連付けを　　*x_range=left.x_range*で行っているところです。これにより、右側のウィンドウは左側のウィンドウでの選択/ズーム/移動によって（またはその逆で）更新されます。

TOOLSリストは、結果表示に含めたいツールのリストに過ぎません。　　*gridplot*を使用することで、2つのグラフを隣り合わせに配置しています。

また、GitHubリポジトリでも、結果の　　HTMLを確認することができます。
GitHubから直接配信するには大きすぎるようなので、ダウンロードする必要があります。

まとめ

このセッションでは、さまざまなソースからデータを取得し、同じグラフにレンダリングする方法を探りました。
また、サンプリング頻度の異なるデータも扱っています（気づかなかったでしょう(^_-)）。
Bokehを使うと、グラフを配布可能なインタラクティブビューで簡単に作成する強力なツールを得られます。
次のいくつかのセッションでは、csp.log、apache/iis
アクセスログ、cconsole.logイベントなど、さらにプロットについて探りたいと思います。
Pythonで処理してほしいデータについて何か提案があれば、遠慮なくコメントに残してください。

あなたの験をシェアしてください！ これは読者との対話で成り立つ学習過程を意図したブログです！

リンク

この記事で使用したすべてのファイルは、　　こちら　にあります。
また、ここで説明された一部のテクニックに基づく@murrayoの新しいeButtons抽出ツールもぜひご覧ください。
https://github.com/murrayo/yape

#Python #ツール　　#パフォーマンス　　#視覚化　#Caché

ソースURL:
https://jp.community.intersystems.com/post/%E3%83%87%E3%83%BC%E3%82%BF%E3%82%B8%E3%83%A3%E3%83%B3%E3%82%B0%E3%83%AB%E3%81%AE%E8%A6%96%E8%A6%9A%E5%8C%96-%E3%83%91%E3%83%BC%E3%83%882-%E3%82%88%E3%82%8A%E5%A4%9A%E3%81%8F%E3%81%AE%E3%82%BD%E3%83%BC%E3%82%B9%E3%81%A7%E3%82%88%E3%82%8A%E5%84%AA%E3%82%8C%E3%81%9F%E5%87%BA%E5%8A%9B%E3%82%92%EF%BC%81