Article
Fabian Haupt · Feb 10, 2017 6m read

Visualizing the data jungle -- Part II. More sources and nicer output!

In last week's discussion we created a simple graph based on the data input from one file. Now, as we all know, sometimes we have multiple different datafiles to parse and correlate. So this week we are going to load additional perfmon data and learn how to plot that into the same graph.
Since we might want to use our generated graphs in reports or on a webpage, we'll also look into ways to export the generated graphs.

Loading windows perfmon data

The perfmon data extracted from standard pbuttons report is a bit of a peculiar data format. On first glance it is a pretty straightforward csv file. The first row contains the column headers, subsequent rows the datapoints. However, for our purposes we will have to do something about the quotes surrounding the value entries. Using the standard approach to parse the file into python we will end up with columns of string objects, which don't work well to graph them.

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

Unlike the in the mgstat file we used in the first part, the header names are in the first row. We want the first column to define our index (this way we don't need to re-index the dataframe as we did last time). Finally we have to parse the first column to actually represent a DateTime. Otherwise we'd end up with a string index. To do that, we define a little helper function to parse the perfmon dates:

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

    return dt
```

The ~~~converters~~~ parameter lets us pass it in as handler for the first column.

After running this, we end up with the perfmon data in a 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
```

Note that the majority of columns is currently an *object*. To convert these columns to a usable format, we'll employ the to_numeric function. While we could use apply to call it on every column, that would mess up our index again. So we'll just plot the data directly while piping it through that.

Plotting

For this run we're interested in plotting the total privileged time of all CPUs. Unfortunately the column numbers are not constant and vary with the number of CPUs and drives. So you'll just have to look through and figure out which column it is. In m y example it's 91:

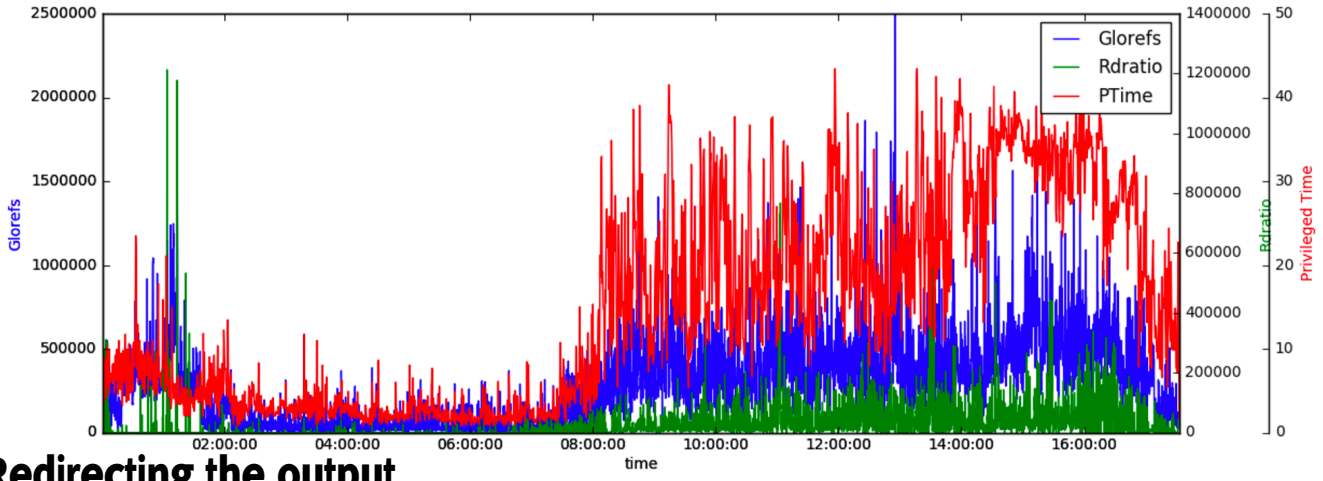
```
perfmon.columns[91]

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

We'll pretty much just use the same approach as last time to create a graph with Glorefs,Rdratio, and our new 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()
```

This is where you get to use *to_numeric*:

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



Redirecting the output

While the notebook is really nice for getting a quick glance of our data,eventually we'd like to be able to run our scripts non-interactively, so we want to output our graphs as images.
Screenshotting is obviously involving too much manual work, so we'll use the pyplot function *savefig()*.

We'll replace the *draw()* and *show()* calls with the *savefig()* call:

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

which will give us the png in our current working directory.

Advanced output

As a little extra exercise we'll have a look at Bokeh. One of the many nice features bokeh is adding to our toolbox, is the ability to output our graphs as an interactive html file. Interactive in this case means, we can scroll and zoom in our data. Add the ability to link graphs together and you can easily create interactive renderings of pbuttons data (or other). These are especially nice, because they run in any modern browser and can easily distributed to multiple people.

For now, we're aiming to just add two graphs to our output. We would like to get *Glorefs* and privileged time from perfom into one page.

For that we'll first need to import bokeh:

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

We'll define a couple of properties and labels as well as the size for each plot. Afterwards we'll just plot the data objects we had collected earlier and we're already done.

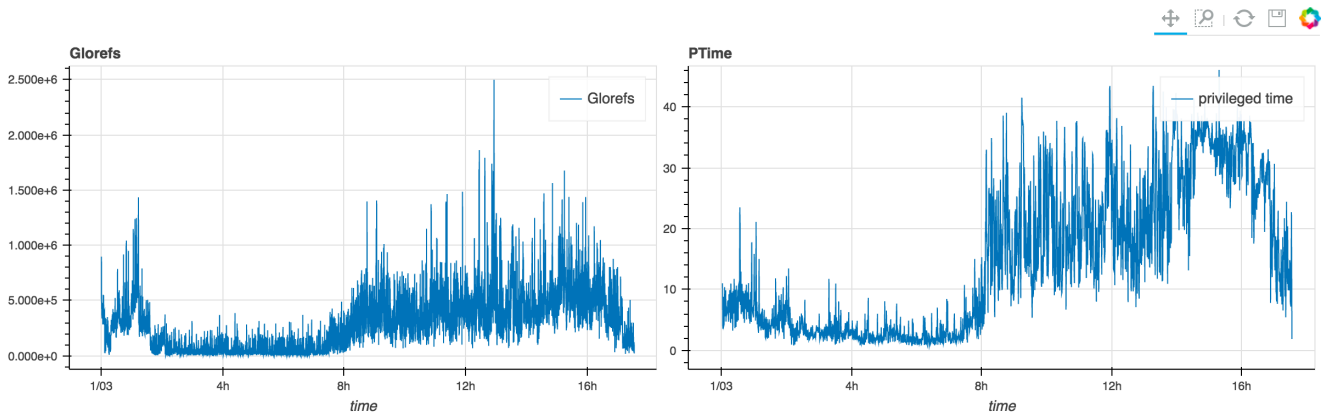
Note the commented out *output_notebook()*, this would render the output directly in our jupyter notebook. We're using *output_file* as we'd like to have a file in the end we can distribute.

```
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)
```

The key ingredient here is the linking of the ranges of our two graphs with *x_range=left.x_range*. This will update the right window with our selection/zoom/move from the left one (and vice versa).

The TOOLS list is just the list of tools we'd like to have in our resulting display. Using *gridplot* let's us put the two graphs next to each other:



You can also have a look at the resulting html in the github repository. It seems to be a bit too big to directly serve through github, so you'll have to download it.

Conclusion

In this session we explored pulling in data from different sources and rendering it into the same graph. We are also handling data with different sampling frequency (bet you didn't notice;)). Bokeh gives us a powerful tool to create easily distributable interactive views for our graphs.
For the next few sessions we'll explore more things to plot: csp.log, apache/iis access logs, cconsole.log events. If you have any suggestion for some data you'd like to see handled with python, please feel free to comment.

Share your experiences! This is very much intended as an interactive journey!

Links

You can find all files for this article on here
Also check out @murrayo's new pButtons extraction tool based on some of the techniques discussed:
https://github.com/murrayo/yape

#Performance #Python #Tools #Visualization #Caché

Source URL: https://community.intersystems.com/post/visualizing-data-jungle-part-ii-more-sources-and-nicer-output