Data Science – System Analyzer (saz)

. ***Srinivasan Viswanathan***

Learning Objectives of the Assignment

1. A data scientist needs to understand the system behavior in terms of cpu, io, memory when processing large amounts of data right from ingesting it from the source to the right storage, and then back for analysis. Data retention policies also become important. In this assignment student will understand those metrics better.
2. Methodology and policies of collecting data – A feel of the issues involved in collecting metrics of any working system be it devices, system, environment via sensors, will be clear. Although the methodology applied here will be extremely simple, projection of that to large scale data collection and ingestion becomes more clear. Students will create a tool for collecting the data and store it in an organized way and later use it for data analysis.
3. Apply pandas/data visualization techniques for anayzing the data. Students will create a tool that will use the data collected in step #2, and will do a spatial and temporal analysis of the data and tabulate the results in various ways. The tool will also have an option to visualize the data.

Pledge of Academic Integrity, honor code (Read this before proceeding further):

1. All the work produced as output should be original output of the student and only that individuals own work.
2. You are welcome to refer to anything to understand the technicalities of the project but at the end when you do the work, please do it out of your own understanding, else go back and repeat.
3. People who are helping or seeking help , please stop at the understaning level and never give the code or take code from others.
4. Students should not post the assignment or source code or solution, in any public or internal forum whatsoever.

Problem

For this assignment you will use Glances a system monitoring framework. (https://glances.readthedocs.io/en/latest/api.html#) You will specifically use the REST API of glance to get the data from the system periodically and store it a local directory of your system. You need to collect enough data (more days better), while you continue to develop your code for the assignment. The following will be the requirements:

Glances Installation and REST API

Install the glances tool and play with it , understand it. For this assignment Use it as a web server without web GUI - *pip install glances; pip install bottle; glances -w –disable-webui* should do it. Collect the following data using REST API: **cpu, percpu, mem, memswap, processcount, processlist, load, diskio, fs, and sensor data**. The API, will give the data in json format. You need to convert the data to csv format and append to the appropriate csv file in the right interval group directory (see below). You will store all collected data in a place where there is atleast 5 GB of space available. This will also be in the config parameters below.

Data Collection

Collect the data using a python program called as psaz_collect.py (program and interface description given below) and store individual data in a csv file. eg., cpu will be in cpu.csv file, percpu will be in percpu.csv, mem in mem.csv, memswap in memswap.csv, processcount in processcount.csv etc. A simple config file will describe the following parameters:

1. data_collection_interval= <value> can be any where from a minimum of 5 seconds to max of 300 seconds.
2. data_directory_isize= <value> Indicates the no. Of intervals of collected data to be stored in ONE directory. Each directory will have the format of “psaz_data.<date>”, where “n” is the interval group number will be monotonically increasing. eg., psaz_data.1 will collect first “i” intervals worth of data and will have one csv file for one group of data for all the “i” intervals. “i” is the data_directory_isize value. If i is 3, and n is 1, first three intervals of data will be in psaz_data.1. And this directory will have all the csv files cpu.csv, mem.csv etc., and each of the files will have 3 entries corresponding to each time interval.
3. data_dir= <string value> Indicates the absolute path where all the collected data will be stored. If this directory for eg., is /psaz then under this directory you will find psaz_data.1, psaz_data.2 etc. Directories and under each of this directories you will find the corresponding csv files.

   The program psaz.py will also store the mapping of the interval group numbers (that will monotonically increase) to the lowest time in epoch (absolute value https://docs.python.org/3/library/time.html). This will help enable saz analyzer queries for the command based on the time intervals to get the data from the right interval group directory(ies) and respond. The mapping file will have entries like the following in each line:
   0:1603062560
   1:1603062681
   2: 1603062800
   ....
   and each entry will have the first column as interval number and the second column following the separator (:) is the epoch value. Basically this should be the time at which you create the new directory corresponding to that interval group. The file should be named as “psaz_map” file.

The program psaz.py can be run simply by *python psaz_collect.py <config-file>* where config-file is the file that contains the parameters described above. Only one such program should be running in the system at any time for obvious reasons. The program will run until you press ctrl-c at which point it will stop. If the program starts running again it will start from where it left off, with a monotonically increasing interval group number (it wil be basically interval group number in the last entry in the psaz_map file plus 1) and start storing them again.

4. data_retention=<value> This will indicate how many group intervals should be retained at any point of time and rest can be deleted. eg., if the value is 100, last 100 group intervals directory will be present and all the old group interval directories will be deleted. This logic could be implemented separately by an independent program that runs once in a while or it could be done as part of the psaz.py program itself. It is up to the student.

Data Analysis

The collected data can be analysed, visualized using the psaz_analyze.py program. It will go through the above data use python pandas package and visualization packages that you went through in the lab classes. The interface for this program is specified as below. The specification below (and above) will collectively drive your design and implementation of psaz_collect.py and psaz_analyze.py programs. Any modules you implement as part of this is internal. Users will only interact with the two programs as specified. Below is the interface specification for psaz_analyze.py program.

The general principle of invoking the analyze program will be the following :
psaz_analyze <category> <optional details> <time granularity options>
where : category – can be any of “cpu”, “mem”, “process”, “sensor”, “load”, “diskio”.

Detail – can be any of the following prefixed with “--” for options to be parsed by the analyze program.
cpu, percpu, mem, memswap, processcount, processlist, load, diskio, fs, and sensor data.
By default psaz_analyse <category> prints the fields listed below as mentioned in the paranthesis for last 30 minutes by default unless the time granularity options specify otherwise.

1. *psaz_analyze.py cpu*
   ctx_switches, idle, interrupts, iowait, irq, syscalls, system, total, user, cpucore
   prints for last thirty minutes as well as the six lines for every five minutes.

2. *psaz_analyze.py percpu*
   cpu_number, idle, iowait, irq, system,total,user
   prints for last thirty minutes. It will print six lines for every five minutes if the option –cpu <corenumber> is specified.

3. *psaz_analyze.py mem*
   active, available, buffers, cached, free, inactive, percent, shared, total, used
   prints for last thirty minutes as well as the six lines for every five minutes.

4. *psaz_analyze.py memswap (average, max (time), min (time))*
   total,used,free,percent,sin,sout
   prints for last thirty minutes as well as the six lines for every five minutes.

5. *psaz_analyze.py processcount (average over last half hour)*
   pid_max,running,sleeping,thread,total
   prints for last thirty minutes as well as the six lines for every five minutes.
6. *psaz_analyze.py processlist (aggregated over last half hour)*
   cmdline, cpu_percent,cpu_times (parent user time, parent system time, child user time, child system time), readio (io_counters[0]-io_counters[2]), writeio (io_counters[1]-io_counters[3]), memory info (rss, vms, shared, text, data), memory_percent, name, num_threads, status (running, sleeping etc.),
   This one prints for all processes. (a variation with –-pid <pid> or –-name <pname> option will list for 30 minutes plus the six lines for each 5 minutes).
7. *psaz_analyze.py load (average over the last half hour)*
   min1, min5, min15, cpu_core
   You could print the average over last half hour and for every five minutes.
8. *psaz_analyze.py diskio(aggregated over the last half hour)*
   disk_name, read_bytes, read_count, write_bytes, write_count
   prints for last thirty minutes, and then gives the six lines of every five min worth of data in the case of –disk_name <name>
9. *psaz_analyze.py fs (actual used for last half hour everything else as is)*
   fstype, mnt_point, begin_percentage, begin_size, begin_free, used (in last half hour), end_size, end_free, end percentage
   gives the percentage used, total size, and free at the beginning of the last half hour, and at the end, and says total used in last half hour.
   And also print the six lines for every 5 minutes in the last 30 minutes in case the option is --mntpoint <name> .
10. *psaz_analyze.py sensors (average over last hour, max(time), min (time))*
    label, type, unit, max (time), min(time), average,

    eg., cpu, 63, C, 60(10min), 40(20min), 50
    where 60 degrees is max, 10 min before in the last 30 min, and 40 is min 20 min before in the last 30 min. It also prints six lines for every 5 minutes.

Time granularity Options

In general for all time options suffix of m will be minute, h will be hour and d will be day.

1. The time detail options are available for all the category of commands described above. "*--last*" 5 min, (minimum) 30 min (default), 1 hour or 1 day. Eg,

   psaz_analyze.py sensors –last 1d

   will give the details of the average for the last one day, and will print one line for every 5 min, as to what the values were.

2. *--granularity* can be 5 min min, till whatever time it wants. eg.,

psaz_analyze.py sensors –last 1d –granularity 1h
will print the same as option #1 above, but instead of granularity of 5 min , it will print 24 lines of the average for each hour including the max/min within that hour in each line.

3. --*start* <time_start> --*end* <time_end>
will print the data similar to –last and –granularity except that instead of last 30 minutes, it will print from start time, to end time. The time format could be "year-month-day:hour:minute" eg.,

psaz_analyze.py diskio –disk_name *dev/sda1* --start 2022-10-06:08:00 –end 2022-10-07:08:00 –granularity 1h

asks for data for a disk with a particular name *dev*/sda1 from october 6 8 am till october 7th 8 am, at granularity of one hour in each line. Now if you do the same with –graph (option below), it will draw a graph that can be saved.

Details Options

These options are for detailing per line based on the particular object, say cpu0, or disk name, or process name or mount point names. eg., below:

1. *psap_analyze.py percpu --cpu 0*

   prints the percpu values for cpu0 only for last 30 minutes and six lines for the every five minute option.

   psap_analyze.py cpu –cpu 0 –last 20m –granularity 10m

   prints the values for cpu 0 for last 20 minutes and two lines one for each 10 min.

2. *psap_analyze.py mem –process <pname>| --pid <pid>*

   The above will print the memory related values for a particular process with name (if there are multiple pids aggregate them) or with a particular pid. The memory related value for a process is rss, vms, shared, text, data etc. It will print one line for the aggregate for last 30 min, and one for every 5 min by default or based on the granularity specified.

3. *psap_analyze.py cpu –process <pname> | --pid <pid>*

   The above will print cpu related values from the process list or for a particular pid similar to the mem above.

4. *psap_analyze.py diskio –disk_name <diskname>*
5. *psap_analyze.py fs –mnt_point <mount point>*
6. The last of the detailed options common for all of the above is *–graph,* that instead of printing in rows format will pop up a graph that can be saved by the user.
7. Final option is –corr that will print the corrrelation coefficient for all the columns printed in any of the above commands. One special command that can be used to correlate is to relate fields from different category as well eg., percpu –field <total> sensors –temperature –cpu 0. This correlation coefficient will be displayed as heatmap between the different fields. And for specialized relationships. For eg., is there a relationship between cpu usage and memory, between cpu usage and temperature of the cpu etc.

Bonus: All of the above are for local system. If we have to centralize the aggregation across many systems, then all of the commands should have –server <hostname> . And have the ability to

aggregate the same data above on a perserver basis. For eg., psap_analyze cpu by default will print the last half hour data for all the servers. And psap_analyze cpu –server <server1> will do exactly wanted above. This just gives an example of the aggregation complexity across multiple servers.

Conclusion

The document is fairly expressive to start with, and we will clarify questions as we go along. In summary the assignment implements a data collection module, (psap_collect), and an analyze module (psap_analyze). As we go through the course we will subject the collected data to some machine learning model mappings if possible and see if we can predict the capacity requirements.