NSF/IUCRC CAC PROJECT

MONITORING, VISUALIZING, AND PREDICTING HEALTH STATUS OF HPC CENTERS

Jie Li

Doctoral Student, TTU

04/24/2020

Advisors:

Mr. Jon Hass, SW Architect, Dell Inc.

Dr. Alan Sill, Managing Director, HPCC, TTU

Dr. Yong Chen, Associate Professor, CS Dept, TTU

Dr. Tommy Dang, Assistant Professor, CS Dept, TTU

LOAD AVERAGE & CPU USAGE

Univa Corporation

Univa Grid Engine Documentation

Univa Grid Engine Administrator's Guide

Author:
Univa Engineering

Version:
8.5.4

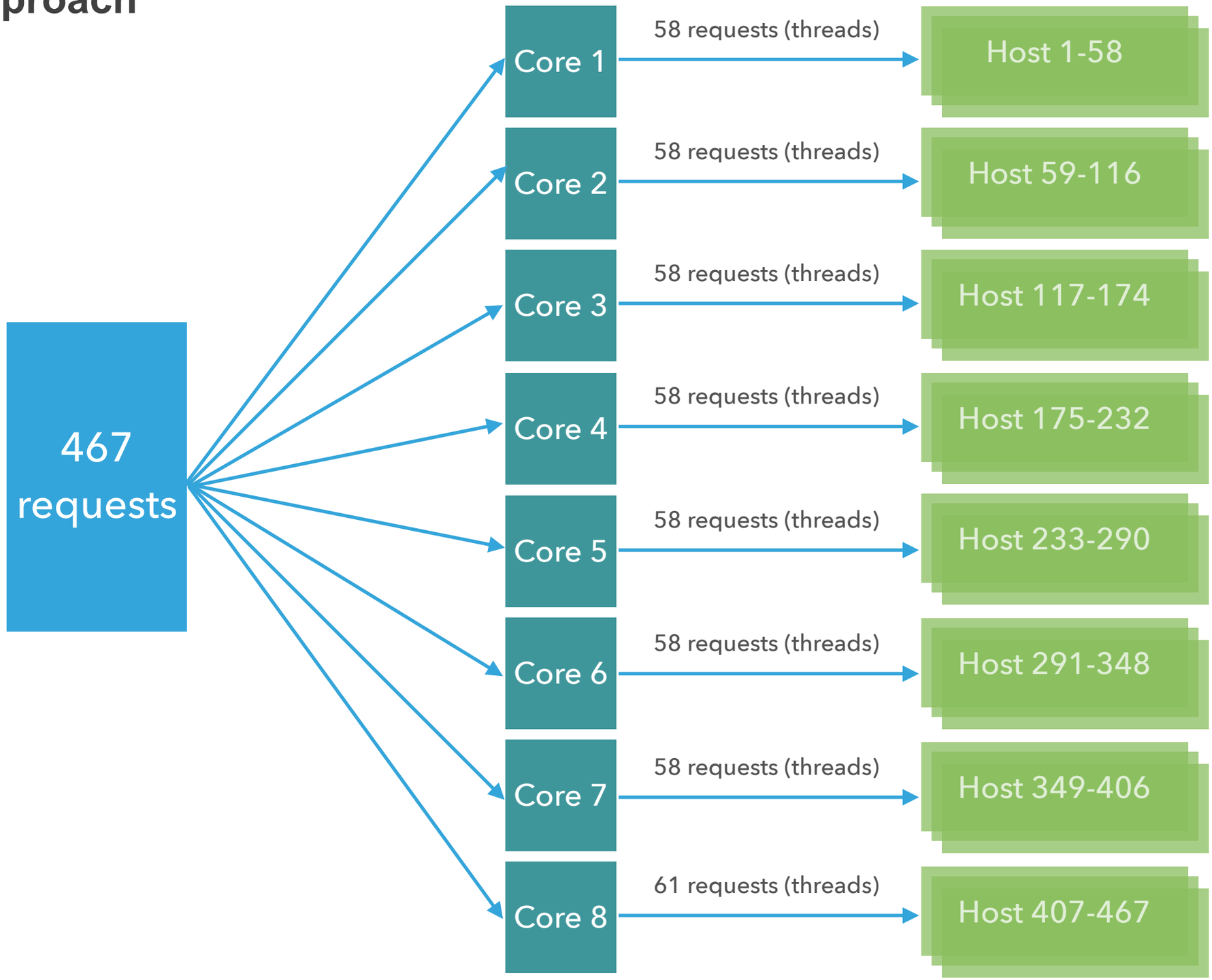
State	Description
swap_total	The total amount of installed swap space.
virtual_total	Total amount of virtual memory (memory + swap space).
load_avg	Same as load_medium.
load_short	Average load value in the last minute (time interval may differ on OS; source on Linux is `/proc/loadavg`).
load_medium	Average load value in the last 5 minutes (time interval may differ on OS; source on Linux is `/proc/loadavg`).
load_long	Average load value in the last 15 minutes (time interval may differ on OS; source on Linux is `/proc/loadavg`).
mem_free	The amount of unused memory.
swap_free	The amount of unused swap space.
virtual_free	The amount of unused virtual memory.
mem_used	The amount of occupied memory.
swap_used	The amount of occupied swap space.
virtual_used	The amount of occupied virtual memory.
cpu	Current amount of CPU usage.
m_topology	Execution host topology information (S means socket, C core, and T hardware supported thread).
m_topology_inuse	Execution host topology like above. Additionally occupied (via core binding) cores are displayed in lower case letters.
m_socket	The number of CPU sockets.
m_core	The total number of CPU cores.
m_thread	The total number of hardware supported threads.
np_load_avg	Medium average divided by number of processors (num_proc).
np_load_short	Short load average divided by the number of processors (num_proc).
np_load_medium	Medium load average divided by the number of processors (num_proc).
np_load_long	Long load average divided by the number of processors (num_proc).
display_win_gui	On Windows (win-x86) only, this value denotes if the execution host is able to display the GUI of a job on the currently visible desktop.

LOAD AVERAGE & CPU USAGE


```
"resourceNumericValues": {
  "num_proc": 72,
  "virtual_total": 302211078816,
  "m_mem_total_n0": "95.909G",
  "m_mem_total": "191.908G",
  "arch": "lx-amd64",
  "mem_used": 85477365383,
  "cpu": 50.1,
  "m_mem_total_n1": "96.000G",
  "m_mem_used": "84.434G",
  "m_topology": "SCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTSCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTT",
  "m_cache_l2": "256.000K",
  "swap_total": 99482179994,
  "m_socket": 2,
  "load_avg": 35.66,
  "m_numa_nodes": "2.000000",
  "m_mem_free_n1": "53.880G",
  "m_mem_free_n0": "50.491G",
  "m_cache_l1": "32.000K",
  "m_core": 36,
  "m_thread": 72,
  "m_topology_inuse": "SCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTSCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTT",
  "np_load_avg": 0.495278,
  "load_medium": 35.66,
  "m_topology_numa": "[SCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTT][SCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTTCTT]",
  "mem_total": 202727825080,
  "m_cache_l3": "45.000M",
  "load_short": 35.97,
  "np_load_short": 0.499583,
  "m_mem_free": "107.475G",
  "docker": 0,
  "swap_used": 20585778250,
  "mem_free": 117250459697,
  "np_load_medium": 0.495278,
  "virtual_free": 196146861441,
  "m_mem_used_n1": "42.120G",
  "virtual_used": 106064217375,
  "swap_free": 78896401744,
  "m_mem_used_n0": "45.418G",
  "load_long": 35.59,
  "np_load_long": 0.494306
},
```

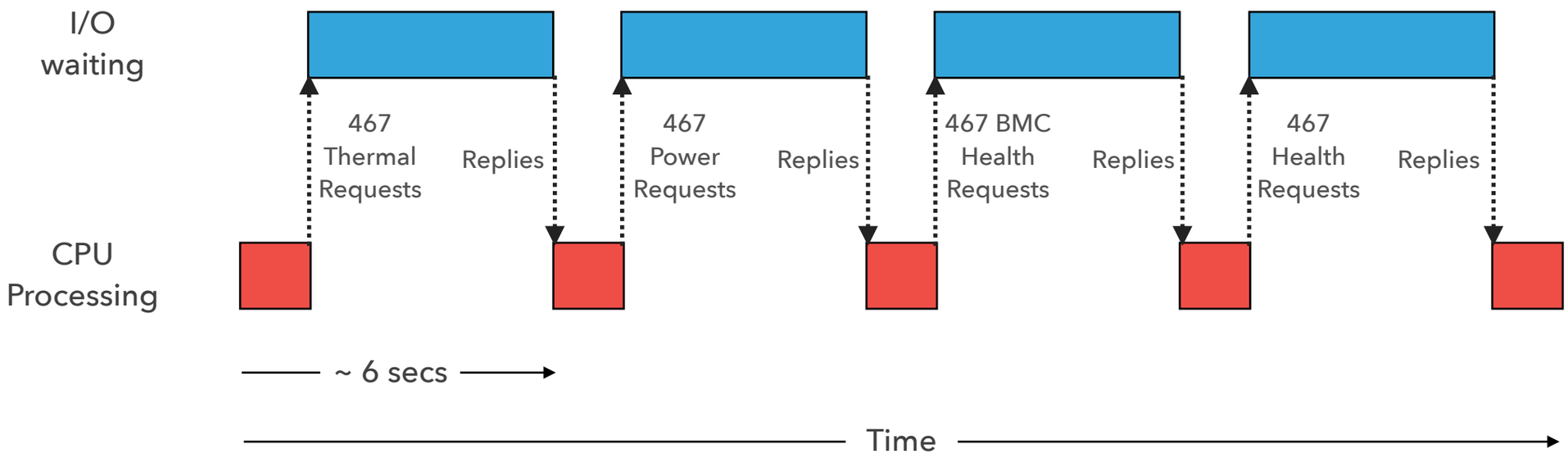
RETHINKING METHODOLOGY OF FETCHING BMC METRICS

Current approach



Ref: Defining Classes of Metrics and Policies for Efficient Data Center Monitoring, Presented by Mr. Ghazanfar Ali, 2/27/2019

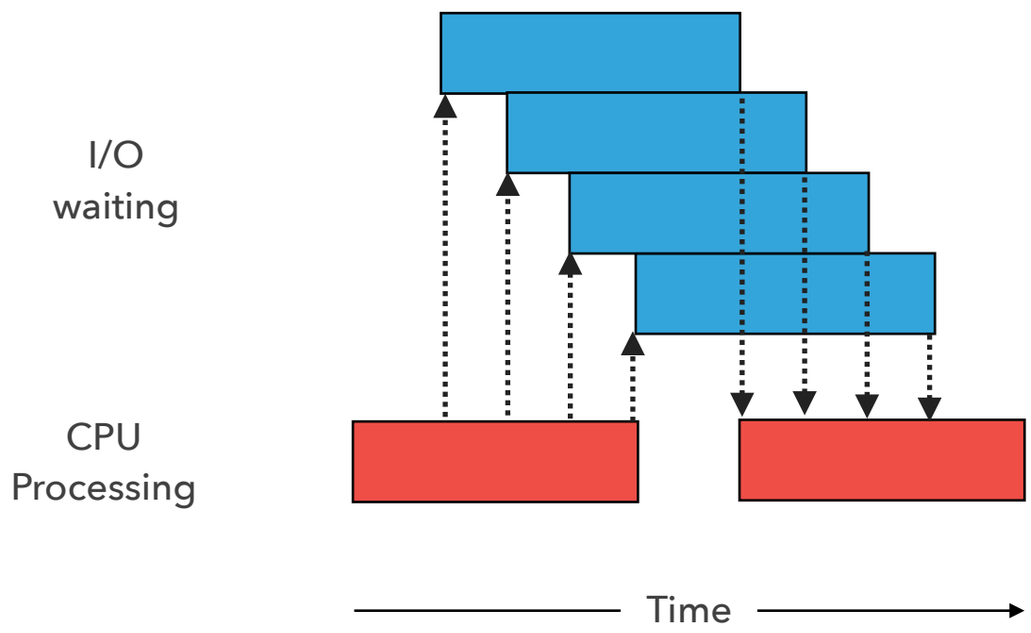
Current approach



- Synchronous, blocks the application while waiting for the server to reply
- Around 58 threads running on the same core, which may degrade the performance because of GIL (Global Interpreter Lock)

https://wiki.python.org/moin/GlobalInterpreterLock

Possible approach

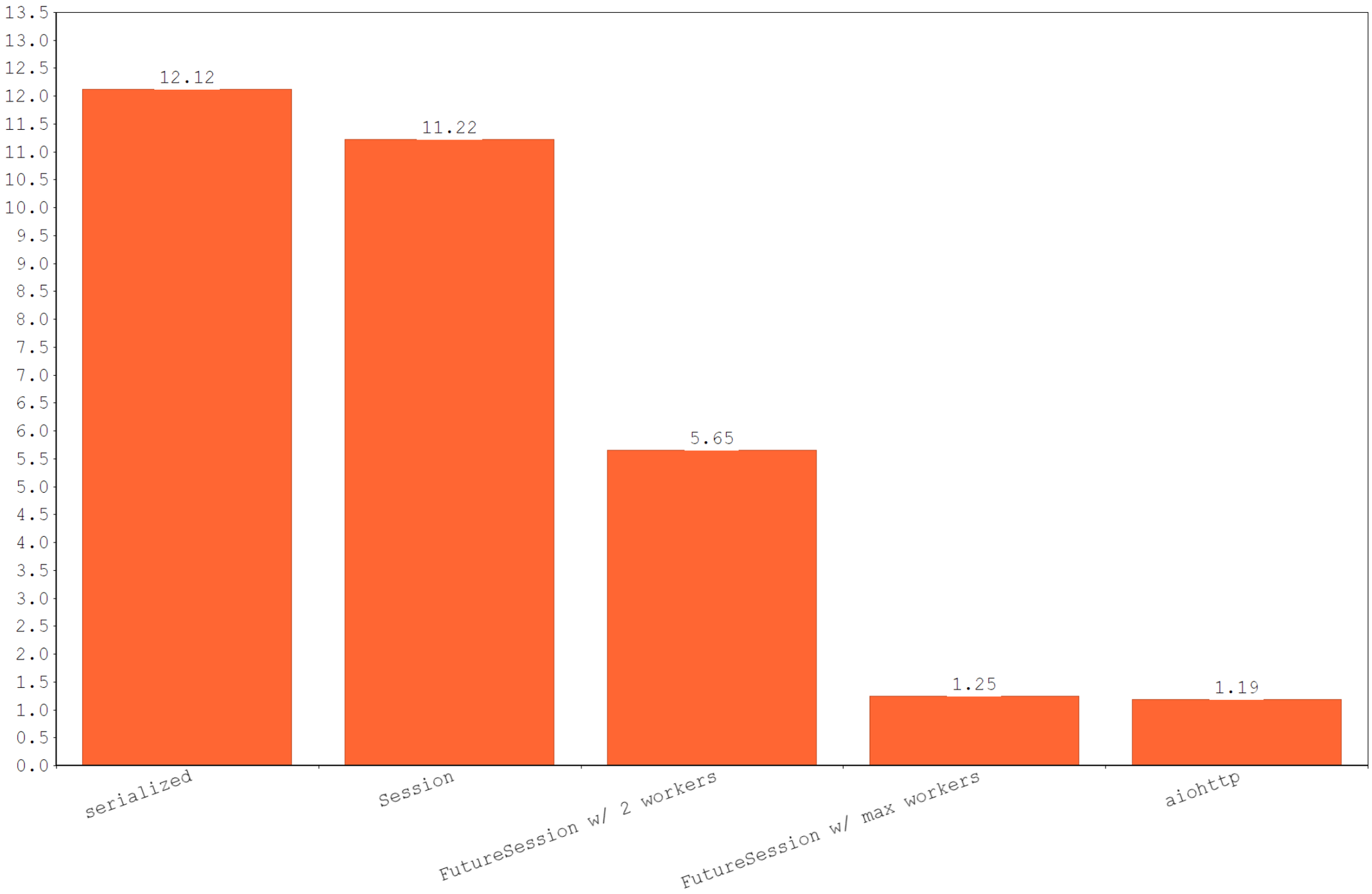


- Asynchronous, sends requests in series but without waiting for the first reply to come back before sending the new one
- Aiohttp, asynchronous http client/server framework, sends the requests over multiple connections in parallel
- Only one thread, does not have GIL issue

https://docs.aiohttp.org/en/stable/

PERFORMANCE

Sending requests to httpbin.org, an HTTP API that provides an endpoint simulating a long request



https://julien.danjou.info/python-and-fast-http-clients/

PROGRESS

- Have implemented the code
- Due to the current collecting script is still running and BMCs may not work well to handle concurrent requests, many failed requests even using retries
- Arrange a time for further testing

QUESTIONS?/COMMENTS?