Preferred Networks Intern Screening 2017 Coding Task for Back-end Development

Changelog

- 2018-5-1: Initial version

Notice

- Please use only the standard library unless otherwise noted.
- Please select from the following programming languages:
  - C, C++, Python, Go, Scala, Rust, Java, C#, Ruby
  - Selecting multiple languages is allowed.
- Do not share or discuss any details of this coding task with anyone.
- Please tackle the task by yourself. Do not share or discuss this coding task with anyone including other applicants. If we find an evidence of leakage, the applicant will be disqualified. If one applicant allows another applicant to copy the answer, both applicants will be disqualified.
- We expect you to spend up to two days for this task. You can submit your work without solving all of the problems. Please do your best without neglecting your coursework.

Things to submit

- Please submit the code that solves the problems and a report describing how to build and run the code.
  - e.g., execution environment, compiler, and other necessary steps

Evaluation

We will evaluate your submission based on the following criteria.

- Whether it outputs correct answers
- Processing time
- Memory consumption

- Readability of the source code
- Coverage with unit tests to ensure the correct behavior of the source code
- How easy it is to reproduce the experimental results
- The submitted report is concise and easy to follow.

How to submit

- Create a zip archive with all of the submission materials and submit it on this form. Due date is Monday, May 14th, 2018, 23:59 JST.

Inquiry

- If you have any question on this problem description, please contact us at intern2018@preferred.jp. An updated version will be shared with all applicants if any change is made. Note that we cannot comment on the approach or give hints to the solution.

Task description

Implement a tool for analyzing log files.

Target files are assumed to be "static", you don't have to deal with online processing or stream processing. Use the two files below for every task.

- log_s.zip (MD5: `7976df39e96fe03a1a351ee95193ffd8`)
- log_l.zip (MD5: `68545fd4ce6a3823dbe56318c6a895df`)

Rename the zip file to an appropriate name, like:

```
$ wget https://preferredjp.box.com/shared/static/mc01xtkcn0qmdgbb1r53uzljeva6e9tq.zip -O log_s.zip
$ wget https://preferredjp.box.com/shared/static/gpmix7flrrl4badutdqwo4hr9q2xnasp.zip -O log_l.zip
```

These files contain one directory per server group, and each directory contains a history of log files as shown below:

```
log_l
  |- server100
      |- app.log.0 <- oldest log file in "server100" directory
```

```
    |- app.log.1
    |- ...
    |- app.log <- newest log file
```

The log files includes the following items:

- Log level: DEBUG / INFO / WARNING / ERROR
- Timestamp: Date, Time, Timezone
- Server name (several servers are included in a same file)
- Process name
- Log message, with one of following types:
    - CPU utilization (0.1=10%)
    - error message and stacktrace
    - other

The log analysis tool should have the functions described below. Please think of an appropriate user interface (parameters etc.) yourself.

Task 1

Output the timestamp with server name when CPU utilization exceeds 95%. The threshold of "95%" should be configurable at runtime, for example as a command line argument.

Task 2

Plot sliding window medians of CPU utilization for all servers. The window size is 5 minutes and the step size is 1 minute.

You are free to decide how to show the plots but it is best to combine all plots using a technique such as a heatmap or wireframe surface etc. You can use external libraries for plotting and image processing.

Example

- Server1 CPU utilization median from 2018-04-01 10:00:00 to 10:05:00 is 50%
- Server1 CPU utilization median from 2018-04-01 10:01:00 to 10:06:00 is 55%
- Server1 CPU utilization median from 2018-04-01 10:02:00 to 10:07:00 is 40%

Then, plot {2018-04-01 10:05:00,50%}, {2018-04-01 10:06:00,55%}, {2018-04-01 10:07:00,40%} on "Server1" axis.

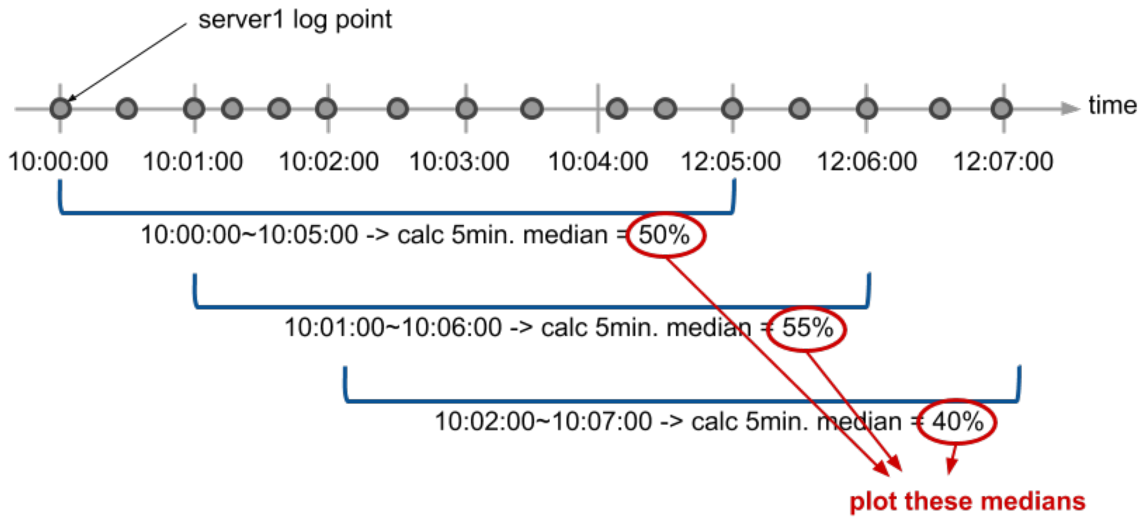
server1 log point
time
10:00:00
10:01:00
10:02:00
10:03:00
10:04:00
12:05:00
12:06:00
12:07:00
10:00:00~10:05:00 -> calc 5min. median = 50%
10:01:00~10:06:00 -> calc 5min. median = 55%
10:02:00~10:07:00 -> calc 5min. median = 40%
plot these medians

Figure 1: how_to_calc_median

Task 3

When medians of CPU utilization calculated in task 2 exceed 90%, show all WARNING and ERROR messages within the 5 minutes window. You can choose yourself how to show these messages.

Example

- Server2 CPU utilization median from 2018-04-01 10:00:00 to 10:05:00 is 33%
- Server2 CPU utilization median from 2018-04-01 10:01:00 to 10:06:00 is 91%
- Server2 CPU utilization median from 2018-04-01 10:02:00 to 10:07:00 is 95%
- Server2 CPU utilization median from 2018-04-01 10:03:00 to 10:08:00 is 88%
- ...

Then, show all WARNING and ERROR messages within [10:01:00–10:06:00] and [10:02:00–10:07:00]. In this case, the range [10:02:00–10:06:00] is overlapping, do not output the same messages twice.

Task 4 (Optional)

This task 4 is optional, please do your best without neglecting your coursework.

Implement an HTTP client to upload the results of task 1 to 3, and an HTTP server that (1) can accept and store the uploaded data and (2) allows to query the data and filter by server name or time range. You can use external libraries and follow any API style, such as REST, GraphQL, or any other.

Example

Create an HTTP server which has an endpoint as described further down, and run it.

```
$ ./my_server start -p 8080
```

Create an HTTP client to register the results which were generated by Task 1 to 3. Please register the results via HTTP, do not store them in the database directly from the client.

```
$ ./my_client register task1_result.txt
$ ./my_client register task2_result.txt
$ ./my_client register task3_result.txt
```

An example to get a list of timestamps when CPU utilization exceeds 95% on “server101” between [2018-04-01 00:00:00 - 00:10:00].

```
$ curl localhost:8080/api/cpu_util/over=95&server=101&from=1522508400&to=1522509000
{
  server: "server101"
  timestamps: [2018-04-01T00:XX:XX.XXX, 2018-04-01T00:YY:YY.YYY, ...]
}
```

An example to get sliding window medians of CPU utilization on “server101” between [2018-04-01 00:00:00 - 00:10:00].

```
$ curl localhost:8080/api/cpu_util/medians_5min/server=101&from=1522508400&to=1522509000
{
  server: "server101"
  medians: [0.1, 0.11, ...]
}
```

An example to get error messages when medians of CPU utilization calculated in task 2 exceed 90% on “server101” between [2018-04-01 00:00:00 - 00:10:00].

```
$ curl localhost:8080/api/logs/level=error&median_over=90&server=101&from=1522508400&to=1522509000
{
  server: "server101"
  logs: [
    {
      time: "2018-04-01T00:XX:XX.XXX",
      level: "ERROR",
      message: "error message"
    }
  ]
}
```

You don’t have to follow these API examples and Output examples