System Evaluation (A1)

System Overview

The system contains two scripts, the cpu_load script collects raw cpu load data from 17dcompv008 via SNMP for a given period of time. The statistics script runs various functions offline to calculate statistics on the given data file. Below displays a list showing the script options and internal functions.

- **cpu_load.sh [seconds]**
    - function *get_cpu_load*
- **statistics.sh [file] [cp95] [cp99]**
    - function *read_measurements*
    - function *variance*
    - function *standard_deviation*
    - function *confidence_interval*

Both scripts have been given executable permissions:

```
chmod +x cpu_load.sh
chmod +x statistics.sh
```

A **data** folder will be created in the same location as the scripts. This is where collected cpu load data files will be stored. The statistics script can then run against these files.

Scripts

cpu_load.sh

The script first sets the total run duration by getting the system *$SECONDS* property plus the specified user seconds (**$1**). Then, at every *tick* (currently set to 5 seconds), it gets the cpu load data.

The cpu load data is from the **UCD-SNMP-MIB::laLoad.1** MIB resource; this is the one minute cpu load average. It then appends this data to the cpu load file. The options specified to *snmpwalk* are *-Oqv*, these enable quick print for easier parsing and prints the value only. This means we only store the raw number.

> **Note:** Each data file is formatted: **[yyyymmdd-hhmmss]-[measureand].txt**, for example *20171111-213148_cpu_load.txt*. The naming convention makes it easier to sort historical runs by date and means we don't need to keep deleting old files or worrying about appending to old files. Also, in coursework part A2, it's easier to group up multiple related data files such as cpu utilisation and memory utilisation per run.

For example, to collect cpu load data over 30 seconds run the cpu load script with the following option:

```
./cpu_load.sh 30
```

statistics.sh

The script first reads from the supplied data file location (**$1**) and stores the results in the *measurements* array. It then calculates various statistics.

The *mean* / central tendency of the measurements is calculated using the below formula:

$$\overline{E[X]} = \frac{1}{N}\sum_{i=1}^{N} xi$$

The *variance* is calculated next, which measures how far the measurements are spread out from the above *mean* value, formula:

$$\overline{E[(X - E[X])^2]} = \frac{1}{N-1}\sum_{i=1}^{N}(xi - \overline{E[X]})^2$$

The *standard deviation* is calculated next which is used to quantify the amount of variation within the data set from the *mean* value. A low value means that most of the measurements are close to the mean while a high value means that the measurements are spread out.

$$\sqrt{\overline{E[(X - E[X])^2]}} = \sqrt{V}ar$$

The *confidence interval* is finally calculated which shows the likelihood that the real value of the cpu load will be in the interval. It calculates the confidence level for both 95% and 99%. The 99% confidence level will have a wider confidence interval. The confidence level values are passed in to the script (**$2** and **$3**)

$$[\overline{E[X]} - \frac{Cp.sd}{\sqrt{N}}, \overline{E[X]} + \frac{Cp.sd}{\sqrt{N}}]$$

The above stats are then printed to the console.

For example, to gather statistics on a specific cpu load data file, run the statistics script with the following options:

```
./statistics.sh /homenfs/pg/b0270122/Postgraduate/data/20171111-213148_cpu_load.txt 1.65 1.96
```

Measurement Procedure & Statistics

The collection of measurements and the statistics are ran independently (the statistics are ran offline from the real system). An improvement to the system would be to concurrently run the statistics script and cpu load script at various intervals to show ‘up to date’ figures. This would however, take some time and the system would need to be adjusted to take this delta time into consideration.

The data received from the MIB endpoint (laload) is set to the 1-minute load average. Using laLoad.2 or laLoad.3 would have given us the 5 and 15 minute load average respectively. Since we’re polling relativity quickly it made sense to use the lower average.

To get the number of available processors, a request to HOST-RESOURCES-MIB::hrProcessorTable was used which returned a list with one item. This information is important when discussing the cpu load chart below.

The cpu_load.sh script is responsible for collecting the actual load data and polls for this data every 5 seconds. The script naively sleeps for 5 seconds on every iteration and therefore does not take in to account the time it takes to do the SNMP request. Thus, in practice each request will actually take a few milliseconds longer than 5 second.

The cpu load data is statistically independent and the statistics script runs various unbiased estimators offline. The detail of each statistic is discussed in the statistics section above.

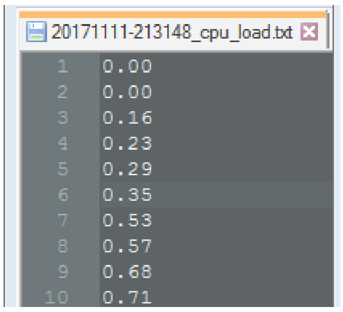
> **Note:** The precision of variables within the statistics script are always at their maximum precision. The precision is rounded to two decimal places for console output only.

Data & Charts

Sample Output

cpu_load.txt

A cpu load data file will contain a list of raw load values, one value per line.


20171111-213148_cpu_load.txt

```
0.00
0.00
0.16
0.23
0.29
0.35
0.53
0.57
0.68
0.71
```

statistics.sh

The script outputs the total number of read measurements and various statistics, explained above. Below is actual output from the statistics script against a cpu load file. The data contained cpu load measurements over a period of 11 hours.

```
Measurements #: 7485
Mean: 0.41
```

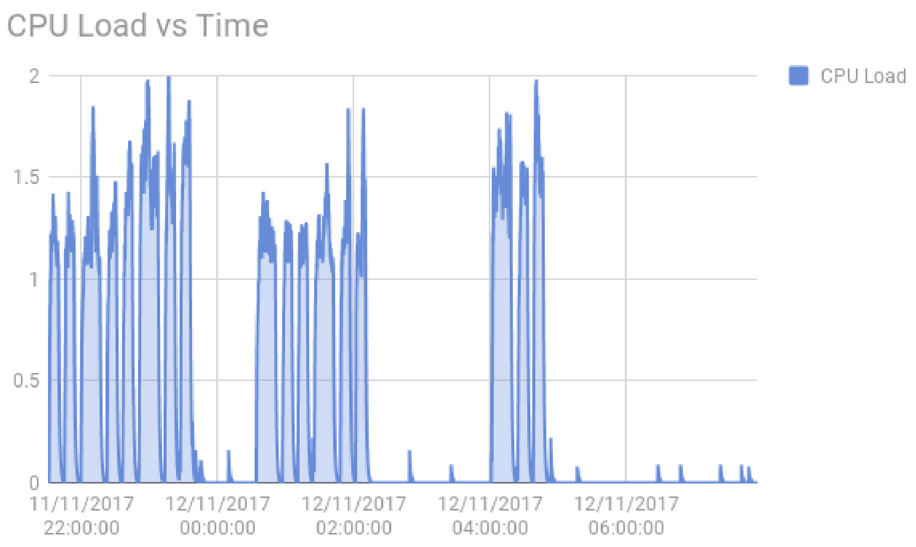
```
Variance: 0.36
Standard deviation: 0.60
95% confidence interval: [0.40, 0.42]
99% confidence interval: [0.39, 0.42]
```

Charts

CPU Load vs Time

The chart below plots cpu load (y-axis) over a period of 11 hours (x-axis). The statistics found are displayed in the output above. It shows various periods of increased load and also troughs within high load periods (around 1.5 -2). There are long periods of increased load, followed by periods of little activity.

As discovered in the measurement procedure section, only 1 processor is available. We can therefore conclude that all values over 1.0 mean that the cpu is overloaded.


CPU Load vs Time
11/11/2017 22:00:00
12/11/2017 00:00:00
12/11/2017 02:00:00
12/11/2017 04:00:00
12/11/2017 06:00:00
CPU Load