P51a Labs

Prof Andrew W. Moore (after Dr Noa Zilberman)

Lent 2022/23

This document provides useful information required for the practicals. All the information is located at: `https://github.com/cucl-srg/P51a`; as we find bugs we will do updates – so `git pull` regularly.

1 Test Machines

Each of you has been allocated a number of test machines, in some places called Machine A and Machine B. These refer to any pair of the machines you have been allocated for Lab1 they may all be treated as identical.

Your machines will be named l51-piXXX.nf.cl.cam.ac.uk where XXX is a nuber from 001 to 060. Each of you is allocated four of these machines. The first machine in each four (001, 005, 009, etc.) is configured with a different extra disk but in all other ways they are identical.

Machines are located in the departments machine-room and are largely self-sustaining. If you have issues raise them with Andrew or Jennifer. If your machine needs rebooting; talk to Andrew.

You will interact with the machines via ssh:

1. I will presume you are using your own laptop; these instructions should also work for the ACS machines in SW02, I generally presume unix (linux) throughout.

2. When you are using your own laptop, you will need to gain access via the departments vpn service. In particular the in house service VPN2 is preferred. Details are at `https://www.cst.cam.ac.uk/local/sys/vpn2`

3. on your machine add the P51a ssh key you have been provided to your ssh keychain, e.g., `ssh-add id_rsa-keyset1`, the keys are available from the restricted area of the moodle; your passphrase will be provided to you in the lab.

4. `ssh -X l51@<hostname>.nf.cl.cam.ac.uk` and enter the password. Hosts ending in .cl.cam.ac.uk or .cst.cam.ac.uk are permitted to ssh into these machines. The `-X` enables X11 forwarding, allowing you to run graphical applications on the

l51 machines from your SW02 machine.

Important: using these machines is a privilege not a right. Please exercise care to ensure you have the correct IP addresses for source and target when doing flood pings in particular as these machines, when misused, can create poor performance in the department networks.

2 Repository Structure

Important: As multiple teams may work on the same machines (using the same `l51` account, clone the repository to a local folder under your crsid, e.g., ∼/awm22/P51a

- `Jupyter`
  - Notebook templates (`.ipynb`)
    For the first lab you will use Jupyter Notebook to run the experiments and save the results.
  - `useful`
    The 'useful' scripts include data processing functions for use in the Notebooks. These are documented later in Section 4.
- `handouts`
  Practical sessions handouts.
- `lab-report`
  A template for lab reports.
- `lectures`
  Lecture slides.
- `setup`
  This folder contains various scripts for setting up tools and creating directories in which to save results.

3 Jupyter Notebooks

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. It is not the only way you will use to gain results but Lab 1 is entirely notebook based.

You will use Jupyter Notebook to run the Lab1 experiments and save the results. The templates are designed to run on Machine A and results are saved to Machine A.
To copy a remote directory onto your local machine, `sftp l51@<hostname>.nf.cl.cam.ac.uk`

and `get -r <directory>`.
Exporting a Notebook as `.tex` will save graphs as separate files, which you can then include in your lab report.

Starting a Jupyter session: after you `ssh -X` into Machine A, type the following:
`cd L50/Jupyter && jupyter notebook`
(`pkill firefox` if Firefox is already running.)
There are several places where you can find more information on using Jupyter:

- Jupyter-Notebook page on the course's wiki.
- guide.ipynb under the handouts folder.
- Jupyter documentation: `https://jupyter-notebook.readthedocs.io/en/latest/notebook.html`
- Jupyter-Notebook Basics page: `http://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Notebook%20Basics.ipynb`

4 'Useful' functions

This is a library of functions for use in several of my modules - not all are relevant to P51a.

4.1 `useful.py`

This script provide useful functions for all practical sessions.

`local_cmd(command)`
Executes Terminal command on local machine. Returns after the command exits.
Input: command, eg. 'echo blah'
Output: Terminal output as a string

`ssh_connect(host)`
Connect to a remote machine using ssh.
Input: IP address of remote machine, eg '128.128.128.128'
Output: paramiko.SSHClient instance (paramiko is a python implementation of ssh v2)

`ssh_cmd(command, ssh)`
Executes a command on remote machine.
Input: command = command. ssh = paramiko.SSHClient instance.
Output: Terminal output (if any).

4.2 useful1.py

This script provide useful functions for Lab 1

`getrtt(ffile,crsid,num)`
Parses a file with RTT results from a ping command.
Input: ffile = '[folder]/[file]', eg. '1/exp1a_0'. crsid - user crsid. num = number of pings.
Output: list of RTTs in microseconds

`graph1(exp,crsid,div,num)`
Plots graph1 in Lab 1.
Input: exp = experiment name, eg. 'exp1a'. crsid - user crsid. div = gaps between error bars. num = number of pings.

`graph5_000001(exp,crsid,usecs,num)`
Plots graph5 for an interval of 0.000001 seconds.
Input: exp = experiment name. crsid - user crsid. usecs = list of rx-usecs. num = number of pings.

`graph5_001(exp,crsid,usecs,num)`
Plots graph5 for an interval of 0.001 seconds.
Input: exp = experiment name. crsid - user crsid. usecs = list of rx-usecs. num = number of pings.

`data_iperf(fexp,crsid)`
Parses the iperf test results.
Input: '[folder]/[exp]', eg. '1/exp1a'. crsid - user crsid.
Output: list of 5 bandwidth values in Gbits/sec for each measurement interval
[[5x bw for 0-1 sec] , ... , [5x bw for 9-10 sec]]

`data10(crsid)`
Parses the data for experiment 10.
Input: crsid - user crsid. Output: tuple containing iperf bandwidth valuess in Gbits/sec of 1. server 2. client
([[5x bw for 0-1sec (ser)] , ... , [5x bw for 9-10sec (ser)]] ,
[[5x bw for 0-1sec (cli)] , ... , [5x bw for 9-10sec (cli)]])

`data11(windows,crsid)`
Parses the data for experiment 10.
Input: windows - list of specified window sizes, in KB . crsid - user crsid.
Output: tuple containing 1. list of actual window sizes in KB 2. list of 5 bandwidth values in Gbits/sec for each window size
([sz1, ... , szn] ,
[[5x bw for sz1] , ... , [5x bw for szn]])

`data_band(fexp,crsid,band)`
Provides a list of effective bandwidth values, based on a parsed results file.
Input: fexp = 'folder/experiment name'. crsid - user crsid.. band = list of effective bandwidth values in Mbits/sec.
Output: tuple containing 1. list of effective bandwidths in Mbits/sec 2. list of 5 percentiles for each bandwidth value.
([bw1, ... , bwn] ,
[[5x % for bw1] , ... , [5x % for szn]])

`data13b(windows,crsid)`
Parses the results of experiment 13b.
Input: windows - list of specified window sizes in KB. crsid - user crsid.
Output: list of 5 bandwidths in Gbits/sec for each window size
[[5x bw for sz1] , ... , [5x bw for szn]]
`graph_error(data)`
Calculates error bars values, for plotting error bars.
Input: list containing n repetitions for each data point.
[[n x point 1], [n x point 2], ... , [n x point m]]
Output: tuple containing 1. list of median y-values 2. correctly formatted yerr for plotting error bars.
([y1, y2, ... , yn] ,
[[below1, below2, ... , belown], [above1, above2, ... , aboven]])

4.3 `useful2.1.py`

This script provide useful functions for Lab 2.1

`gettimes(exp)`
Parses a trace file and extracts a list of timestamps.
Input: experiment name, eg. 'exp1a'.
Output: list of timestamps in seconds as strings.

`getdiff(exp)`
Parses a trace file and provides a list of time gaps between timestamps.
Input: experiment name
Output: list of timestamp differences in nanoseconds

4.4 useful2.2.py

This script provide useful functions for Lab 2.2

`gettimes(exp)`
Parses an trace file and extracts a list of timestamps.
Input: experiment name, eg. 'exp1a'
Output: list of timestamps in seconds as strings

`getdiff(exp)`
Parses a trace file and provides a list of time gaps between timestamps.
Input: experiment name
Output: list of timestamp differences in microseconds

`getrtt(fname)`
Parses a trace file and provides a list of round trip times.
Input: filename Prints examples of RTTs.
Output: list of RTTs in microseconds.

4.5 useful3.py

`getdeltas(fexp, num)`
Parses a trace file and provides a list of inter arrival times.
Input: exp = 'folder/experiment name', eg. '3.1/exp1a'. num = number of packets sent
Output: list of inter-arrival times in microseconds (float)