Toolkit for Doing Wide-area Experiment on PlanetLab Testbed

User Manual

MNG Group
March 31, 2003

1. Introduction

PlanetLab is a project initialized by tens of universities (mainly are universities in USA and Europe) to build a wide area testbed which provides the capacity of doing experiments in a real distributed environment. Currently there are totally about 100 sites and more than two hundred PlanetLab machines, each of them is a Dell PC running LinuxRedHat (kernel version 2.4.19). The details information about PlanetLab can be found at the following websites:

http://www.planet-lab.org/
http://ccwp7.in2p3.fr/mapcenter/planetlab/

SSH is the only way to access to PlanetLab hosts, that implies you would be asked to type password or passphrase (in our case, it is passphrase). We cannot update the configurations like "*/etc/hosts.equiv*", "*/etc/hosts.allow*" and so on to make all PlanetLab machines are friendly or equivalent to each other, that provides the way to access from one PlanetLab machines to another without typing password.

To solve this problem, we implemented a toolkit that consists of a set of Expect scripts. Expect is a kind of script language that works on top of TCL. The main advantage of Expect is that it can automatically finish the process of interaction when people log into a remote machine using SSH. To learn Expect language, please refer to the book "Exploring Expect".

This manual presents a brief description for installing and using the toolkit. The establishment of the related experiment environment is discussed too.

2. Getting Started

2.1 Supported Platforms

The current toolkit implementation is designed to work on a Linux PC, in which Expect is integrated as a part of standard distribution. This toolkit has been tested on RedHat 7.1 (kernel 2.4.2-2), RedHat 7.3 (kernel 2.4.18-3) and RedHat 8.0 (kernel 2.4.18).

2.2 Set Up Your PlanetLab Account and Password/Passphrase

Before starting work on PlanetLab testbed, you need to build an account and set up password/passphrase. This password/passphrase is stored in a private/public key pair. You need to upload your public key when creating the account, and keep your private key on your local host. According to the policy of PlanetLab, there are ten login accounts (like uv1, uv2, …, uv10) managed at each PlanetLab site. The administrator at the local site will assign a login name (such as uv1) to your account. After the authentication, your public key will be distributed over all PlanetLab hosts, and you can use the assigned login name and your private key (with your password/passphrase) to login to any PlanetLab host or start a remote job on a remote PlanetLab host. The PlanetLab toolkit serves to automate the interaction with all PlanetLab hosts.

The following steps shows the procedure to establish a PlanetLab account:

Set up your account

Before accessing PlanetLab hosts, you should establish your account and accept authentication from your local administrator. The following steps are needed to set up your account:

- **Register on the PlanetLab testbed:**

  Go to PlanenLab homepage, click "Users" which is at the left side of the page. On the displayed page, click "new account" and follow the instruction to fill the registration form. Click "register" and finish the registration. Don't forget the passphrass you typed while generating your private-public key pair.

  Note: The following command is used to create a public/private key pair:

  ```
  ssh-keygen -t rsa -f ~/.ssh/identity
  ```

  It will ask you to type a passphrase, DON'T FORGET IT!. The output is two files: identity and identity.pub. Be careful when you create a private-public key pair, because the key generate command will overwrite the key files in the directory HOME-DIRECTORY/.ssh, here HOME-DIRECTORY is the home directory when you login to a local machine using your account.

  Note: Make sure the public key file consists of a single line. On some operating systems (e.g., Windows, Solaris), the ssh-keygen program generates public keys that contain newlines and causes a problem. We recommend you to create the public/private key pair on a Linux host.

- **Enable your account:**

  Send e-mail to local administrator, including the account name and E-mail address, to tell that your PlanetLab account has been created and needs to be enabled. The local administrator will enable your account and assign a login name (for example uv3) to your account. It usually takes one or two hours for the login name to become available for you to use after your PlanetLab account is enabled.

Send e-mail to local administrator, including your name and E-mail address you typed in the registration form, to tell that your PlanetLab account has been created and needs to be enabled. The local administrator will enable your account and assign a login name (for example uv3) to your account, and inform you through e-mail. It usually takes one or two hours for the login name to become available for you to use after your PlanetLab account is enabled.

- **Ensure your private key is stored in the right place:**

Make sure that your private key (it is the file *identity*) in the local directory: HOME-DIRECTORY/.ssh. It is usually done by the key generate command.

2.3 Downloading and installing PlanetLab Toolkit

- Go to the PlanetLab toolkit website
  http://www.cs.virginia.edu/~mngroup/software/PlanetLab_toolkit/
  and download *planetLabtools.tar.gz.*
- Save the downloaded package into a directory DIR, execute the following command:

```
cd DIR
tar –zxvf planetLabtools.tar.gz
```

A sudirectory named *planetLabtools* will be created under the directory DIR. The directory *DIR/planetLabtools* is your working directory.

2.4 Prepare working environment

- Configure the settings in the configuration file: *planetLab.conf*

```
cd DIR/planetLabtools
vi planetLab.conf
```

The contents of the file *planetLab.conf* look like:

```
UserID=my_login
Passphrase=my_passphrase
Working_directory=my_working_dir
```

You need to replace "*my_login*" with your actually login name, for example, *uv3*, and replace "*my_passphrase*" with the passphrase you typed when creating private/public key pair. For security reason, we recommend you to type in your passpgrase in the file planetLab.conf before starting your experiments and delete it from the file planetLab.conf when the experiments are finished.

The “*Working_directory*” is used for all commands with prefix “scp”. When specifying “*my_working_dir*” as the working directory, the actual remote directory is REMOTE_HOME_DIR/my_working_dir. For example, if set “*Working_directory=widearea-test*”, then the command “*scp_command_unify host-list.txt 0 10 /home/hypercast/tmp/test.txt*” will copy the local file /home/hypercast/tmp/test.txt to the directory REMOTE_HOME_DIR/widearea-test on the remote hosts specified by “host-list 0 10” (explained in the User Manual).

- Prepare host list file. Most PlanetLab toolkit scripts work with a host list file, which is a list of current active PlanetLab machines. Any line beginning with "#" is comment line. A sample host list file, host-total.txt, is provided with PlanetLab toolkit package. The full list of PlanetLab hosts can be found at: https://www.planet-lab.org/db/nodes/nodelists.php.

  Note: The status of PlanetLab testbed is not very stable due to the dynamics of Internet. Before starting your experiment, you need to test PlanetLab hosts (using ssh_test_alive or ssh_test_alive_unify command) and make sure the hosts you used are alive. A failed host should be commented by inserting "#" at the beginning of the line.

- When you want to run your program on remote PlanetLab hosts, you need to prepare the working environment first, including installing needed software, such as JAVA, and setting PATH environment variable on remote hosts. When multiple persons share a login name, each one should create and work in his or her own working directory.

Now you are ready to use the PlanetLab toolkit.

3. Scripts for Running Experiments on PlanetLab Hosts

In this section, we describe the functions of the scripts (commands) in the toolkit. Except the script ssh_login, all other scripts in the toolkit are working with a text file in which multiple PlanetLab hosts are listed randomly or in the order depending on the requirement of your experiment. The host list file, for example, hostslist-file, looks like:

```
#128.111.52.61
150.135.65.2
128.2.198.199
129.237.123.250
#150.135.65.3
#128.95.219.192
```

Blank lines and the comment lines (begin with #) will be ignored by all scripts.

1. ssh_login

`Usage: ssh_login hostname`

This command automatically logins in remote host *hostname* without typing passphrase.

2. **ssh_login_batch**

```
Usage: ssh-login_batch hostlist-file
```

When typing command "*ssh_login_batch hostlist-file*", you first automatically login into the machine listed as the first one in the file *hostlist-file*. Type "exit" under the shell prompt will exit the current connected machine and automatically login in the next machine. The process will continue until all machines have been accessed.

3. **scp_command**

```
Usage: scp-command hostlist-file sentfile-file
```

The file *hostlist-file* is specified in the same way as command ssh_login_batch. *Sentfile-file* is the full name of the local file you want to copy to remote machines. In the above script, for each remote host, the local file *sentfile-file* will be copied to the remote directory REMOTE_HOME_DIR/$Working_directory that is defined in the configuration file *planetLab.conf*. The time limit on executing the scp command for each remote host is 1800 seconds (30 minutes). It is enough to finish most scp commands. An example of scp_command is shown as below:

```
scp_command  hostlist-file  /home/hypercast/hypercast.prop
```

4. **scp_fromremote**

```
Usage: scp_fromremote hostlist-file remote-file-name local-dir
```

This script copies a file from remote machines specified in the file *hostlist-file* to local directory *local-dir*. It is very useful for collecting data files from multiple remote machines, for example, typing

```
scp_fromremote hostlist-file *udp.txt /home/hypercast/test/
```

will copy files **udp.txt* from remote directory REMOTE_HOME_DIR/ $Working_directory to local directory /home/hypercast/test.

5. **ssh_command**

```
Usage: ssh-command hostlist-file command-file|command
```

The file hostlist-file is as before, the command, which will be executed on all remote machines speficied in the file *hostlist-file*, can be stored in the file *command-file* or specified directly in the command line. Here are two examples:

Example 1: ssh_command hostlist-file "rm ./widearea-test/*udp.txt". It will remove all files matching **udp.txt* under the remote directory REMOTE_HOME_DIR/widearea-test.

Example 2: ssh_command hostlist-file command-file, the contents of the file command-file are:

```
    source .bash_profile;cd ./widearea-test;java  -classpath
         UnicastTest.jar udp_r 2000
```

It will start JAVA program *udp_r* on all specified remote hosts.

Note: In most experiments, we need to run non-interactive tasks on remote machines, there are some tricks to do that with Expect scripts. The above script is a way to run remote non-interactive jobs (please check the contents of the script), it works. But it may not be the best way. If you find some better ways, please let me know.

Note: If you prefer to observe the output of the command on each remote host, you need to edit the script ssh_command by replacing the following line:

```
expect -re ".*" {puts "\nfinished!\n"}
```

with line:

```
expect eof
```

6. ssh_test_alive

```
Usage: ssh-test_alive hostlist-file
```

The script ssh_test_alive is used to test whether the machines specified in *hostlist-list* are alive or dead. Note that the command we use to test a remote machine runs a non-interactive task on remote machine, which will not display any message on local screen to show the results. In this case, the only way to find a down machine is timeout of the command which implies we don't get any response from the tested remote machine.

Note: If the script *ssh_test_alive* tells a host is down, this host may not be really down. This host may be alive but just responses after a long time. You can try script *ssh_login* to detect whether this host is really down. On the other hand, if the script ssh_test_alive says a host is alive, this host is certainly alive.

The above scripts will execute identical commands on all hosts specified in the file *hostlist-file*. To provide a flexible way to only execute commands on a selected set of hosts, we define the following set of variants of above scripts.

7. ssh_login_batch_unify

```
Usage: ssh_login_batch_unify hostlist-file start_index num_of_hosts
```

8. scp_command_unify

```
Usage: scp_command_unify hostlist-file start_index num_of_hosts sentfile-
       file
```

9. scp_fromremote_unify

```
Usage: scp_fromremote_unify hostlist-file start_index num_of_hosts remote-
       file-name local-dir
```

10. ssh_command_unify

```
Usage: ssh_command_unify hostlist-file start_index num_of_hosts command-
       file|command
```

11. ssh_test_alive_unify

```
Usage: ssh_test_alive_unify hostlist-file start_index num_of_hosts
```

The above derived set of scripts do the same work as their original versions, but only work on the set of hosts defined in the file *hostlist-file* that starts at the host with index *start_index* and includes *num_of_hosts hosts*.

4. Customize the Scripts in the Toolkit for Specific Objectives

The scripts in the toolkit usually execute the same action on each remote host. Sometimes this behavior doesn't satisfy our requirement. For example, on each remote host, we create a file *.buddylist* to cache the logical coordinates of the remote host. When you want to copy the *.buddylist* file on a set of remote hosts to a local directory, the scripts described in the section 3 cannot correctly finish this job. It is easy to update standard scripts to do this specific work. The only thing needs to be done is to edit the script *scp_fromremote* (or *scp_fromremote_unify*) by replacing the following line:

```
    spawn scp "$UserID@$host:/home/$UserID/$WorkingDirectory/$sentfile" $dir
with
    spawn scp "$UserID@$host:/home/$UserID/$WorkingDirectory/$sentfile"
            $dir/$host.coord
```

We give the updated script a new name: *scp_buddylist_fromremote* (or *scp_buddylist_fromremote_unify*). We typing the following command:

```
scp_buddylist_fromremote hostlist-file .buddylist /home/hypercast/test/
```

For each remote host (assume its IP address is *REMOTE_IP*) specified in the file *hostlist-file*, it will copy the *.buddylist* file from the remote directory REMOTE_HOME_DIR/ $Working_directory to the local directory /home/hypercast/test/ and store it in the file *REMOTEIP.coord*

In this toolkit, we create a few of customized scripts for specific goals. You can create your own customized scripts in the same way.

12. scp_buddylist_fromremote

```
Usage: scp_buddylist_fromremote hostlist-file remote-file-name local-dir
```

13. scp_buddylist_fromremote_unify

```
Usage: scp_buddylist_fromremote_unify hostlist-file start_index num_of_hosts
          remote-file-name local-dir
```

The meaning of the scripts *scp_buddylist_fromremote* and *scp_buddylist_fromremote* are explained above.

14. scp_buddylist_toremote

```
Usage: scp_buddylist_toremote hostlist-file remote-file-name local-dir
```

15. scp_buddylist_toremote_unify

```
Usage: scp_buddylist_toremote_unify hostlist-file start_index num_of_hosts
          remote-file-name local-dir
```

For a host (assume its IP address is REMOTE_IP) defined by "*hostlist-file*" or "*hostlist-file start_index num_of_hosts*", either of the above two scripts copies the file $local-dir/ REMOTE_IP.coord to the remote file: REMOTE_HOME_DIR/ $Working_directory/.buddylist.

There are multiple other customized scripts in the toolkit, including: *ssh_coord_first*, *ssh_coord_first_unify*, *ssh_coord_follow*, *ssh_coord_follow_unify*, *ssh_ttcp_tcp*, *ssh_ttcp_tcp_unify*, *ssh_ttcp_udp* and *ssh_ttcp_udp_unify*. They are created by the similar method mentioned above. If interested, you can take a look of the script to understand what these scripts do. It is an easy job.

5. Extend the PlanetLab Toolkit for working in Other SSH-oriented Environments

The interaction processes of accessing different remote hosts using SSH are similar, with the minor difference. For example, When running a SSH command to access a PlanetLab host, you are asked for a passphrase. Specifically the following message is displayed on the screen:

Enter passphrase for key '/home/hypercast/.ssh/identity':

The scripts in the toolkit are customized to satisfy this requirement. But other operation systems might ask for a password. The message displayed on the screen looks like:

hypercast@iodine's password:

To adapt to the new requirements of the different system, you only need to make a minor update in the scripts. In each scripts, replace the following line:

```
        expect -re "Enter passphrase for key '.*':"
with
        expect -re "password:"
```

6. Doing Experiments on PlanetLab Machines

6.1 Create Command Files

For the script ssh_command and its variants, the command(s) that will be executed on remote hosts can be stored in a text file, called command file. This is useful when the remote commands are too long and complex to type each time. In this toolkit, several command files have been created:

- command_killJava
  ```
  killall -9 java
  ```
- command_RunServer
  ```
  source .bash_profile; cd ./widearea-test; java  -classpath
       hypercast2.0.jar:parser.jar:jaxp.jar
       edu.virginia.cs.mng.hypercast.testing_and_monitoring.RS2 ""
       128.143.71.37/1500 1
  ```
- command_runClientProxy
  ```
  source .bash_profile; cd streaming_xa/clientProxy.xa; ./runp.sh
  ```
- command_runServerTv
  ```
  . .bash_profile; cd streaming_xa/serverProxy.xa_tv; ./run.sh
  ```
- command_runServerUnt
  ```
  . .bash_profile; cd streaming_xa/serverProxy.xa_untouchable; ./run.sh
  ```

You can create your own command file in the same way. Creating command files can save a lot of time and avoid unnecessary typing errors.

Note: The commands in a command file should be written in a single line separated with semi-colon signs.

6.2 An Example of Experiment

Step 1. Setup DT server. Execute the following command on a local machine where DT server is running:

```
java -classpath hypercast2.0.jar
edu.virginia.cs.mng.hypercast.DT.DT_Server 8081
```

Step 2. Start RunControl. Execute the following command on a local machine where RunControl is running:

```
java -classpath hypercast2.0.jar:parser.jar:jaxp.jar
edu.virginia.cs.mng.hypercast.DT.DT_RC2 noserver 1500
```

where 1500 is the port number for connecting RunServers.

Step 3. Start RunServers. Execute the following script on a local machine, for example iodine.cs.virginia.edu.

```
ssh_command_unify hostslist-file 0 20 command_RunServer
```

The contents of the file command_RunServer look like:

```
source .bash_profile; cd ./widearea-test; java  -classpath
hypercast2.0.jar:parser.jar:jaxp.jar
edu.virginia.cs.mng.hypercast.testing_and_monitoring.RS2 ""
128.143.71.28/1500 1
```

Step 4. Type RunControl commands and finish experiment. When RunControl starts, it provides a UI for users to type commands. The following is an example of commands sequence which send UDP packets from the selected sender to all other overlay members.

```
>create_experiment 60
>slow_start_experiment 5
>wait_until_stable
>set_value 0 BWTest-0.Measurement-0.BlocksXBlockSizeXRate 1024x1024x1000
>get_performance
```

Here command "get_performance", which collects the time that all receivers received the last packet, is not included in Hypercast2.0 released version. Commend "set_value 0 BWTest-0.Measurement-0.BlocksXBlockSizeXRate 1024x1024x1000" is also a updated command which sends UDP packets from the sender at the specified rate, it is 1M in this example. In hypercast2.0 standard version, the sender in the command "set_value" sends UDP packets to the network as soon as possible. In wide range network environment, like Internet, it doesn't work well, usually the loss ratio of UDP packets for all receivers is very high that makes the statistics on the delay measurement useless.