Explanation of data structure design

We have created three data structures to store data from the Rakefile for our c client:

1) DATA structure will be used to store the hostnames as follows:
    1) type - type will be HOSTS.
    2) value - value hold the name of the different hosts.
    3) busyflag - busyflag will have the value 0 if hostname is free, otherwise 1 if busy.

    A Rakefile having the following line HOSTS = hostname1 hostname2 will be stored as follows:
    HOSTS hostname2 0
    HOSTS hostname1 0

    Please note that initially all remote hosts servers will be set to free, i.e busyflag will be 0.

2) ACT structure will be used to store the different actionsets and their supporting commands as follows:
    1) act_num - act_num is the actionset number.
    2) action_number - action_number refers the action number irrespective of the actionset number.
    3) place – place will be 1 for local actions and 0 for remote actions.
    4) value – value will store the command to be executed.

    A Rakefile having the following lines will be stored as follows:
    actionset1:
        ls
        remote-echo hello
            requires a.txt b.txt
    actionset2:
        echo helloagain
        remote-ls
            required c.txt d.txt
    1 1 1 ls
    1 2 0 echo hello
    2 3 1 echo helloagain
    2 4 0 ls

3) FILES structure will be used to store the required files for an action as follows:
    1) action_number - action_number refers the action number irrespective of the actionset number.
    2) files_required – all the files required by the action having action_number

        A Rakefile having the lines as shown above will be stored as follows:
        2 a.txt b.txt
        4 c.txt d.txt

For the python client, we have stored the port number as an integer variable, the host servers will be stored in a dictionary and the action data will be stored in nested lists.

Usage

If file named Rakefile is present in the current directory, no command line argument is needed. Otherwise, pass the file path as a command line argument.

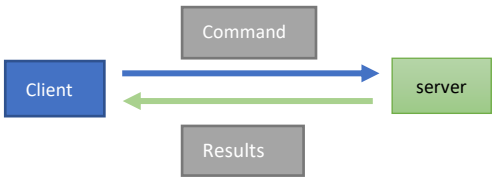
The hostname in Rakefile should include the domain name. (Syntax example: hostname.domain)

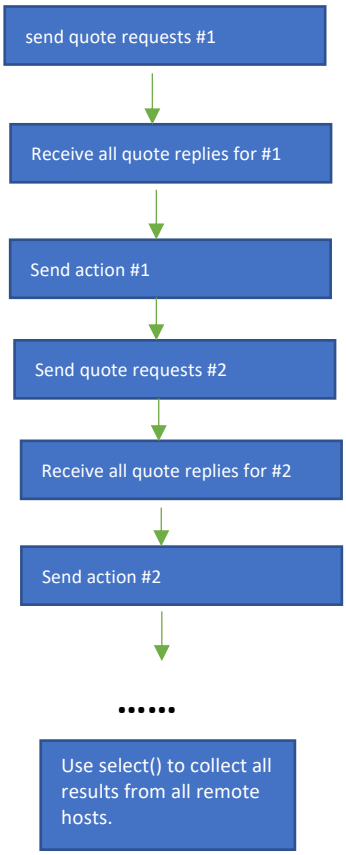
Protocol designed for communication between client and server

- Socket programming has been used for creating connections between servers and clients.
- popen() has been used to execute the different commands on local and remote servers.
- select() has been used to collect/receive results from remote hosts.
- rand() has been used to stimulate the cost of executing an action on a particular server.
- Clients will only send commands and the files required to server. Server execute commands and return the results back to the client.

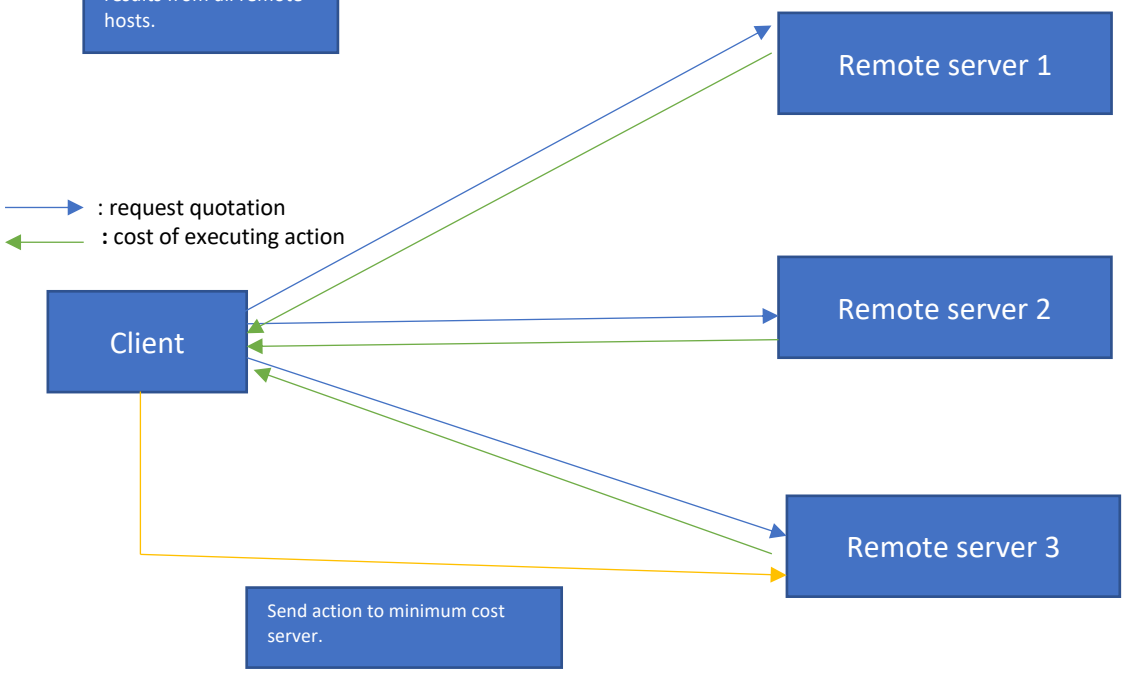
We have one client and multiple instances of servers. Actions of the first actionset will be executed first. If all actions in that particular actionset succeeds, only then we move on to executing the actions of the next actionset.
For each action within an actionset, we will check whether the action is to be executed locally or remotely.

For local action, we will send the command action directly to the local server. The local server will execute the commands and send back the results and exit status back to the client (If the execution fails, send exits status and any output sent to stderr back to client).





Otherwise, for remote action, we will first create connections to all free (check busyflag) remote servers and request each remote free server to provide a quotation on a command. Each server will then stimulate the cost of executing the action and will send the cost back to the client. The client will then determine which server replied with the minimum cost. The client will then send the action along with any required files to this particular server. The remote server will try to execute the command and send back the output to the client. (blocked by select())



If there are no free servers, select() will be used to get all the action results. All servers will then be set to free.

The above process will be performed for each action in a particular actionset.

Execution sequence to compile and link multi-file program

If a particular action in an actionset requires files, the client will try to open those files. If some of the files cannot be opened, the client will throw an error and will proceed to the execution of the other actions in the actionset. In the case where some of the files cannot be opened, the next actionset will not be executed. If all of the files can be opened, then the client will send all the files necessary to execute that action to the lowest cost server together with the filename of each file. Once the server has received all the files, the client will then send the command to be executed to the server.

When does remote compilation and linking perform better?

- If we do not have all the resources or environment necessary to compile the files locally on our machines, remote compilation and linking would enable us to efficiently finish the task without wasting time and resources on our computer. (Installing other environments ect…)
- If there are too many files that need to be compiled at the same time, compiling those files on different servers will save much time and resources compared to compiling those files locally.