Design Document

Parsing

1. The command received from the command line is of the form $node2.\,cmd1 \mid node5.\,cmd2 \mid node3.\,cmd3 \mid \ldots$ .Such input is split into tokens using the ‘|’ character as a delimiter using the *tokenise_string* function. Each token after this processing step has the form $nodeX.\,cmd$
2. The *get_actual_cmd* function is used next to retrieve the node information and command to be executed from the tokens.
3. (on the server) *tokenise_strin*g function is utilized again to retrieve the argument list that is to be passed to the *exec()* function call.

Reading the config file and initializing the required globals

1. The config file should have the format such that each line has a comma-separated name and IP address mapping.
2. Each line in the config file corresponds to a unique node.
3. When the client starts its execution, it fetches all the IPV4 addresses associated with its local machine’s network interfaces and saves them in a global array.
4. The IP addresses read from the config file matched against the local IP address information to identify the node’s name on which the client program is executing.
5. Nodes’ information (name, IP) is also stored in a global array of *node* structure.

Handling Connections

1. The node information extracted from the token in the parsing step is used to connect to the correct node.
2. A non-blocking connect with a timeout of 2 seconds is used to establish the connection.
3. After the connection establishes, a 3 step process sends the command and inputs and retrieves the outputs.
    a. The client sends the command to the server.
    b. The client sends the input data to the server.
    c. The server executes the command using the input data and returns the output to the client.

   NOTE: The input and output at each step are truncated to 4kB, keeping in mind the PIPE_BUF size and the limit on the message size that can be put in a message queue.
   Limiting the size to 4KB has a significant impact on reducing the complexity of client and server programs.
4. If a command is to be executed on the local machine, the client forks a child process that executes the command.
   This implementation allows the client to run programs on a local machine even if the Cluster Shell server on that node is offline.

Multiple Piping- Design details

1. The client handles multiple piping by making connections with the nodes iteratively.

2. If the first command in a multipipe input is to be executed on the local machine, then the first command takes its input from stdin; otherwise, the output obtained from the previous command is used.

Broadcasting: Design details

1. A command should be given a prefix of *'n*.'* to execute on all nodes,
   For example, a command n*.ls executes the command 'ls' on all the nodes.
2. For broadcasting, the client sets up a message queue and spawns N children (N->number of nodes). Every child establishes a connection with a unique node and writes the output to the message queue.
3. The client process reads all the messages from the message queue and displays the output on the console.

Special Commands:

1. *nodes* - the client creates a message queue and spawns N children. Each child tries to establish a connection with a node and informs the client if it was able to establish a connection (indicating that the node is active) through the message queue.
   Timeout for connection -2 seconds
2. *clustertop* - Running a clustertop command requires the broadcast of 2 commands to every node.
   a. free- the free command gives detail about the memory utilization by a system.
   b. iostat -Gives information on the CPU utilization. **(requires installation of sysstat package)**

Miscellaneous:

1. The server handles the connection by allotting a thread to each of the connections. The thread detaches itself before executing anything so that its status doesn't have to be reclaimed.
2. To ensure that all the nodes execute commands in the client's user's home directory, The client process retrieves its home directory and does a broadcast to change the working directory of all the active nodes.