Design Document

Requirements:

- To design a monitoring system, which monitors 1000 servers. Each server has 2 CPUs. Each server generates a log for CPU usage every minute.
  The format is required to be designed like:

  ```
  timestamp    IP            cpu_id usage
  1414689783   192.168.1.10  0      87
  1414689783   192.168.1.10  1      90
  1414689783   192.168.1.10  1      93
  ```

  - The simulator should generate the logs for one day, say 2014-10-31. Random numbers between 0% to 100% can be used as CPU usage. The generator will write data to files in a directory. The timestamp is Unix time.
  - A command line tool has to be created which takes a directory of data files as a parameter and lets you query CPU usage for a specific CPU in a given time period. It should be an interactive command line tool which reads a user's commands from stdin.
  - The tool may take several minutes to initialize, but the query result should be returned within 1 second.
  - The tool should support two commands:
    - One command will print results to stdout. Its syntax is *QUERY IP cpu_id time_start time_end*. Time_start and time_end should be specified in the format *YYYY-MM-DD HH:MM* where YYYY is a four digit year, MM is a two digit month (i.e., 01 to 12), DD is the day of the month (i.e., 01 to 31), HH is the hour of the day, and MM is the minute of an hour.
    - The second command to support is EXIT. It will exit the tool.

Example working as per given requirements:

To run the generator:

```
./generate.sh DATA_PATH
```

To run the interactive query tool:

```
./query.sh DATA_PATH
>QUERY 192.168.1.10 1 2014-10-31 00:00 2014-10-31 00:05
CPU1 usage on 192.168.1.10:
(2014-10-31 00:00, 90%), (2014-10-31 00:01, 89%), (2014-10-31 00:02, 87%), (2014-10-31
00:03,  94%) (2014-10-31 00:04, 88%)
>EXIT
```

Assumptions

- The server IP can be taken at random, thus the system is taking server IP from 192.168.1.0 and generating 1000 IPs, corresponding to which data will be generated.
- The monitoring system when generating logs will take System date and generate logs for the given system date. It is also assumed that only one day's logs will be kept in the Log files and they will be in ascending order(this assumption has been taken because the system is generating logs every minute and they will be appended to the existing logs, thus by default they will be in ascending order according to unix timestamp)
- The input time will be given as *YYYY-MM-DD HH:MM* where YYYY is a four digit year, MM is a two digit month (i.e., 01 to 12), DD is the day of the month (i.e., 01 to 31), HH is the hour of the day(in 24 hour format from 00 to 23), and MM is the minute of an hour(in 60 minute format from 00 to 59).
- The monitoring system will generate logs by picking a number from 2700 to 3000. This assumption has been taken because there are 1440 minutes in a day, and if each server writes a log every minute then taking in consideration 2 CPU's in a server and assuming that sometimes a CPU's log is not written every minute or a CPU's log is written more than one times every minute. Thus the random number generation is done within the range 2700 to 3000.
- The unix timestamp is generated at random by calculating the System date's earliest timestamp, at 00:00 and at random the chosen number of logs are generated from the range of 00:00 to 23:59 this is done internally by the system and does not affects the functioning of the system.
- It is assumed that multiple directories can be created in the given data path. This assumption is taken into consideration so as to keep logs for different servers into different directories. Also, no directory with the same name as predefined servers are already present in the specified data path.

Functionality
According to the given requirement the log generator and query are to be done separately. However, it would make more sense if both of the process run by the same script and work consecutively. This approach is also taken into consideration because this is a log monitoring simulator. Thus, effort has been spent into designing and developing both approaches and the user can use the approach which suits them. Both the approaches take the same assumption and do the same functioning internally. The requirement specified by the user does gives an added advantage that a specified server directory can be queried on, however since the logs in the directory are not being cached or stored in the system memory and log file has to be read every time and the server IP will be needed in the query, thus this is not a bigger advantage. The alternative approach requires to be only initialized once by the system and once the completion the user can directly start writing queries on the system. Both the approaches have been discussed in the following sections.

Approach 1:

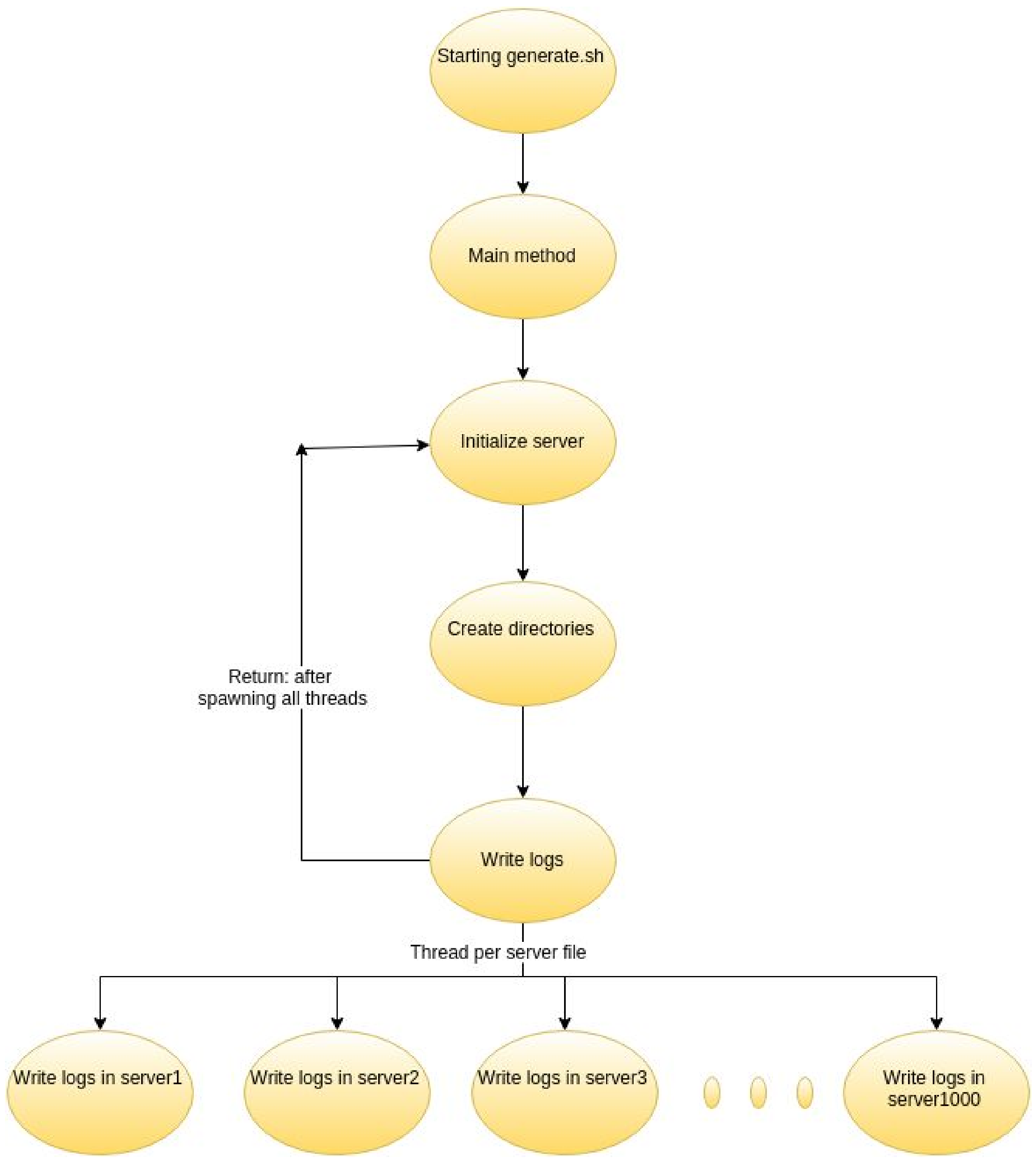
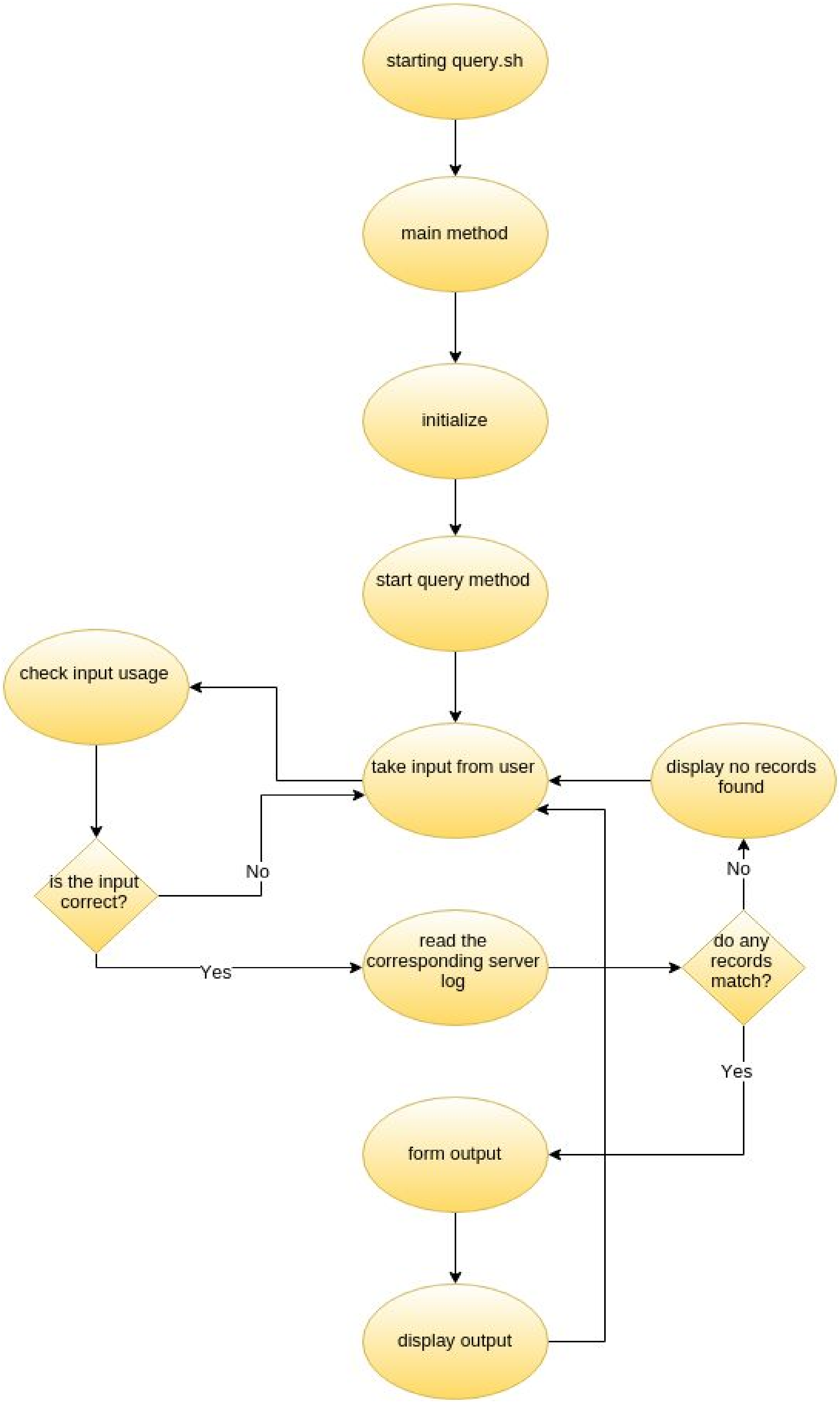


Fig 1. log generation

Fig 2. query simulation(next page)


starting query.sh
main method
initialize
start query method
check input usage
take input from user
display no records found
is the input correct?
No
Yes
read the corresponding server log
do any records match?
No
Yes
form output
display output

Functioning:

Log generation

- generate.sh takes a data path from the user.
- If no data path from the user has been given or if multiple command line arguments have been passed, then an error is thrown.
- Once the command line arguments are verified, then initialisation starts and 1000 pre-defined server IP's are created.
- The data path is validated. If the data path has been entered incorrectly, the user will need to restart the system and enter the correct datapath.
- After the validation of data path, 1000 sub directories are created in the given data path, corresponding to predefined server IP's. The log files will be stored in these data paths.
- The directory creation proceeds to writing off logs. For writing of logs into servers, threads have been used. A thread is spawn corresponding to every server IP and that thread writes the logs into a server log file and then dies when log writing has been completed.
  This process has been depicted in *figure1*

log monitoring via query

- query.sh takes a data path from the user
- If no data path from the user has been given or if multiple command line arguments have been passed, then an error is thrown.
- Once the command line arguments are verified, then initialisation starts and 1000 pre-defined server IP's are created, which are similar to the IP's created in the log generation process.
- The data path is validated. If the data path has been entered incorrectly, the user will need to restart the system and enter the correct datapath.
- After the validation of data path, the user will be instructed of the query format, and then can start writing queries.
- Once the user has entered a query, the system will verify if all the desired parameters have been entered correctly or not.
- If all the parameters have been entered correctly, then the user proceeds to file the corresponding log file. If the corresponding log file is missing, then an exception is thrown. This would mean that the log file has been removed by the system or it has been deleted.
- Once the corresponding log file has been found, the system will start to read the file and find the start time in it which had been entered by the user. This is done by translating the input time into unix time and then checking it in the log file. This all is done internally by the system.
- When the start time is found, then the system starts to reads logs from the start time till the end time and forms an output according to the given requirements.
- Once the system has read all logs between the time period then it proceeds to display the found logs.
- If no logs have been found in the given time period, it displays no logs found.

- Once the process has been completed for the given query, the system goes back to the state for accepting another query from the user.
- If the user wants to exit the system, then '*exit'* command needs to be entered and the system will exit.
  This process has been depicted in *figure2*

Approach 2(Alternative approach):
We saw that some of the tasks are being repeated in *generate.sh* and *query.sh*, thus it would also make sense if those repeated tasks can be eliminated. This would make the system faster and more efficient. To achieve this goal, the following alternative approach has also been implemented.

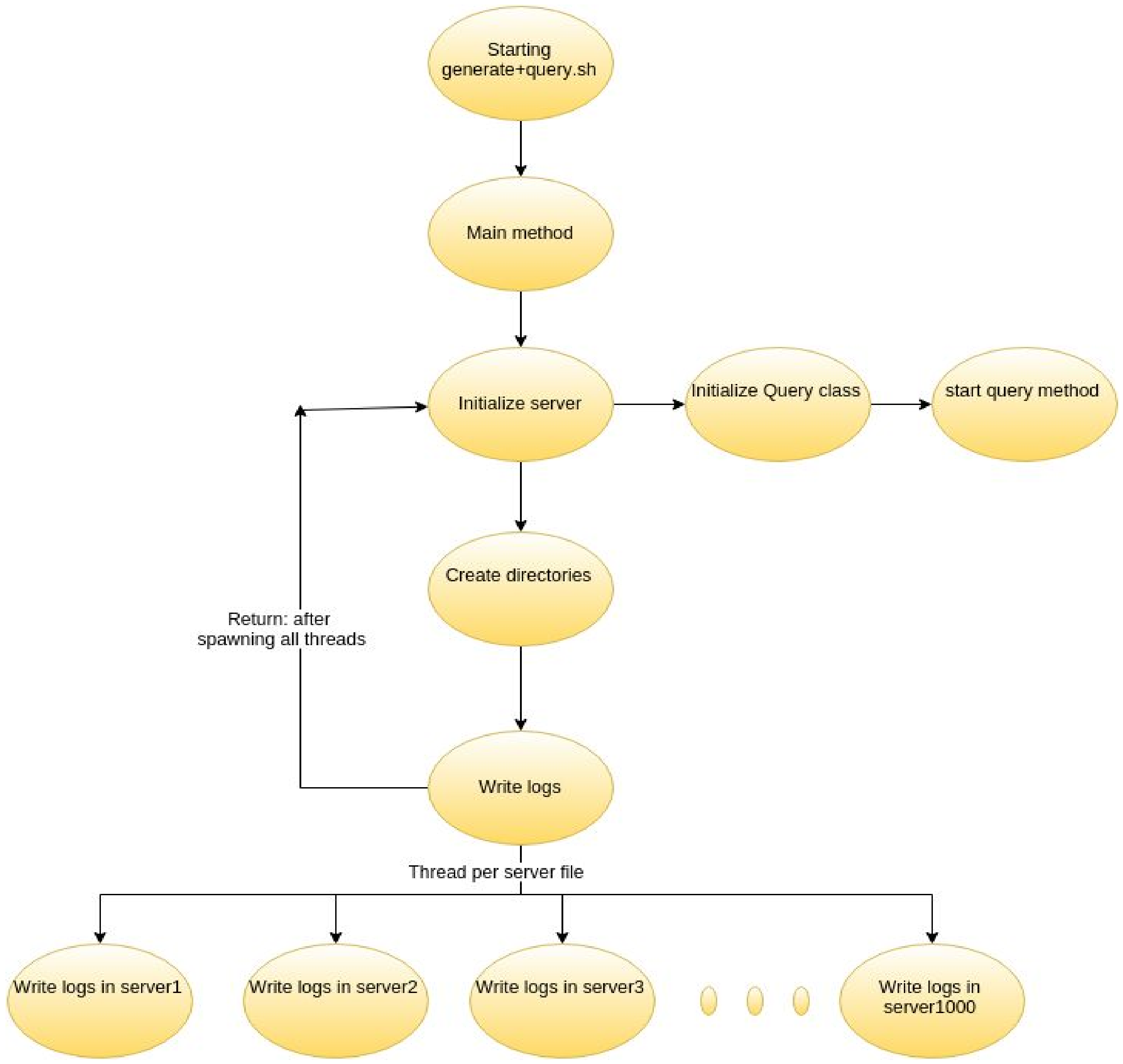


Fig 3. generate+query simulation

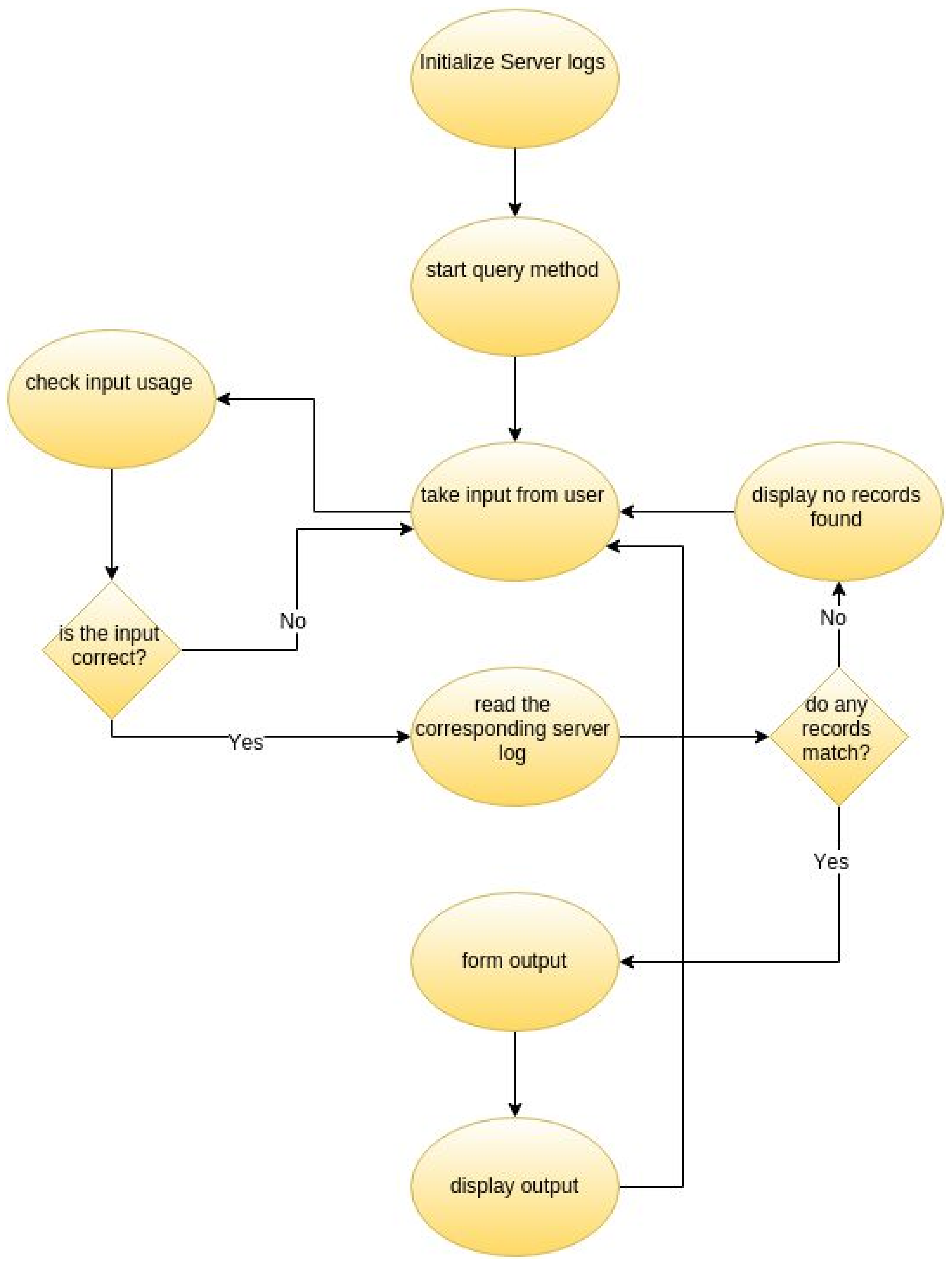


Fig 4. generate+query simulation

Functioning:

Log generation and monitoring via query

- generate.sh takes a data path from the user.
- If no data path from the user has been given or if multiple command line arguments have been passed, then an error is thrown.
- Once the command line arguments are verified, then initialisation starts and 1000 pre-defined server IP's are created.
- The data path is validated. If the data path has been entered incorrectly, the user will need to restart the system and enter the correct datapath.
- After the validation of data path, 1000 sub directories are created in the given data path, corresponding to predefined server IP's. The log files will be stored in these data paths.
- The directory creation proceeds to writing off logs. For writing of logs into servers, threads have been used. A thread is spawn corresponding to every server IP and that thread writes the logs into a server log file and then dies when log writing has been completed.
- Once all the threads have been spawned for log generation, the system returns to intialize method and create an object of the Query class passing in the data path and list of defined server IP's. This removes the repetition of creating list of server IP's and validating command line parameters and validating the data path, and the user can directly proceed to writing queries to monitor the logs.
- Now the user can start writing queries.
- The format for writing a query is given to the user.
- Once the user has entered a query, the system will verify if all the desired parameters have been entered correctly or not.
- If all the parameters have been entered correctly, then the user proceeds to file the corresponding log file. If the corresponding log file is missing, then an exception is thrown. This would mean that the log file has been removed by the system or it has been deleted.
- Once the corresponding log file has been found, the system will start to read the file and find the start time in it which had been entered by the user. This is done by translating the input time into unix time and then checking it in the log file. This all is done internally by the system.
- When the start time is found, then the system starts to reads logs from the start time till the end time and forms an output according to the given requirements.
- Once the system has read all logs between the time period then it proceeds to display the found logs.
- If no logs have been found in the given time period, it displays no logs found.
- Once the process has been completed for the given query, the system goes back to the state for accepting another query from the user.
- If the user wants to exit the system, then '*exit'* command needs to be entered and the system will exit.

Testing

- Each individual module has been tested individually and each function has been tested as well.

Conclusion
The given system has been designed and developed according to the given requirements. An alternative approach has also been developed keeping in consideration that the system needs to be fast and efficient.