System Design

The multi-cloud model that we consider in this work is comprised of the following main components:
• A set of clouds (MCE) = {C1*; C2; : : : ;* Cm}, where each cloud has a set of service files F = {F1*; F2; : : : ; Ff* } and each file is composed of a set of services S = {S1*; S2; : : : ;* Ss};
• A service requester, which is a user interface that accepts a user's request and then displays the SC sequence formed;
• A similarity scaler to measure how similar the received request is to previously received requests;

• A cloud combiner to select suitable clouds (those with the highest number of matching services that fulfill the request) and then generate the combination list based on the suitable clouds; and
• A composer to receive the combination list of clouds from the combiner and then select the services that best fulfill the request from each cloud. Then, the composer produces the SC sequence.

Our MCE simulator is implemented in a Java-based tool that uses client/server programming based on sockets to model the interactions in an MCE at different scales.

the main components of the simulator:
• **Generator**: The generator tool is responsible for the production of the configuration file, which stores the number of clouds in the MCE, their properties, and the available services.
• **Configuration editor**: The configuration editor is a thread that periodically changes the configuration file to reflect the dynamic nature of the cloud environment.
• **Emulator**: The emulator reads the configuration file and uses threading to produce the clouds needed on their corresponding IP address and port. The produced clouds are accessed by the service client through a TCP connection that retrieves a list of the cloud's services or calls a specific service in order to view its contents.

Service client: The service client is a library used for service requests by our MultiCuckoo algorithm and all the other benchmark algorithms we implemented.

Performance Evaluation

Simulating an MCE at different scales by:

1) Varying the number of clouds (NoC) in the environment at 10, 15, and 20 clouds. Due to limitations of the capabilities of our hardware, we were not able to experiment with a larger number of clouds.
2) Varying the average number of services per cloud (NoS) at 15, 25, and 35 services.
3) For each scenario, randomly generating user's requests for combined services to ensure representative samples of the possible number of combined services and clouds per request.
4) Running each algorithm with different user requests of different lengths.
5) Running the MultiCuckoo algorithm with different similarity thresholds and finding the most appropriate threshold.
6) Measuring the performance of the MultiCuckoo algorithm using:
   a) total number of examined services,
   b) number of combined clouds, and
   c) running time.