Prototyping Assignment Report

TU Berlin, Fog Computing SS 2024

Prepared by Group 9:
Melisa AKBAYDAR, 490649
Felix ZAILSKAS, 487751

Requirements & Our Fog Architecture

In the prototyping assignment, it was required to develop an application with components that run both on a local machine (the Edge) and in the Cloud (Google Cloud Engine, GCE). The Edge must collect and utilize simulated environmental data from at least two virtual sensors, generating realistic data continuously. This data must be exchanged regularly between the Edge and the Cloud, with multiple transmissions occurring per minute in both directions. In the event of a disconnection or crash, both the Edge and Cloud components must continue to operate independently, saving data locally for later transmission. Once the connection is restored, any queued data must be transmitted to ensure synchronization.

Considering these requirements, we have developed a smart AC model that aligns with the required Fog architecture, as shown in Figure 1.

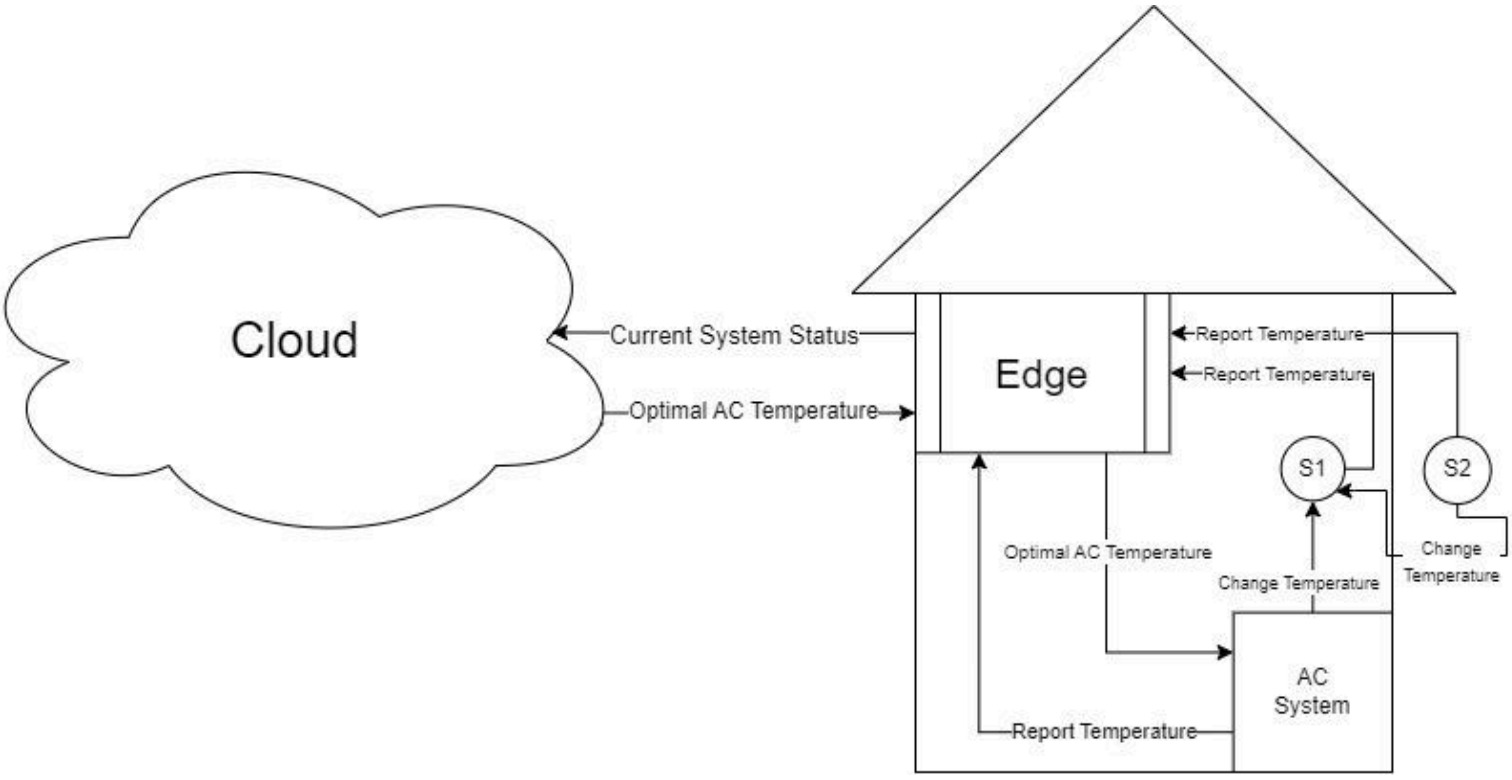

Figure 1

The Smart AC system in the house uses two temperature sensors: one inside the home and one outside. These sensors measure and collect temperature data at a time span. As the outside temperature fluctuates randomly, it impacts the inside temperature. The goal is to maintain the room temperature at a desired level by adjusting the AC. To maintain a comfortable living environment, the inside temperature should be kept at an optimal level, which in this case is 21 degrees Celsius.

The temperature data from both sensors is collected every second, accumulated every five seconds, and sent to the cloud server, which calculates the optimal AC temperature and sends it back to the edge client for adjustment. The sensors send their temperature data to the cloud via the Edge. The Edge is shown in the figure in this way to provide high level abstraction to make it seem more separated and easy to understand. In the provided code, you can see this part as “client”. In the code, this component is referred to as the “client”. The AC unit is connected to the cloud and can only be adjusted by the cloud. Using the data received from the Edge, the cloud calculates the optimal temperature for the AC to achieve. This adjustment temperature is then sent back to the Edge using TCP. After the AC is adjusted, the inside sensor should read a temperature of 21 degrees or a value which is close to it. This ensures that the indoor environment remains comfortable despite changes in the outside temperature.

We have created an adjustment function:

$$(\text{Goal Temperature} \times (1 + \alpha + \beta)) - \alpha \times \text{outside\_avg} - \text{inside\_avg} / \beta$$

In this function, α and β are constants representing external factors such as wall thickness, AC material, etc . Our goal is to calculate the adjustment temperature for the AC, which will bring the room temperature close to 21 degrees.

If the difference between the current room temperature and the target temperature is significantly higher than expected, an if-else block is used to expedite heating or cooling of the room.

```python
def compute_ac_adjustment(
    inside_avg: float,
    outside_avg: float,
    ac_temp: float,
    goal_temp: float,
) -> float:
    if inside_avg < goal_temp - 1:
        if outside_avg < goal_temp:
            ac_temp += 2
        else:
            ac_temp += 1
    elif inside_avg > goal_temp + 1:
        if outside_avg < goal_temp:
            ac_temp -= 1
        else:
            ac_temp -= 2

    # Cap the max difference between ac and goal temperature
    threshold = 5
    if abs(goal_temp - ac_temp) > threshold:
        if goal_temp - ac_temp < 0:
            ac_temp = goal_temp + threshold
        else:
            ac_temp = goal_temp - threshold

    return ac_temp
```

This code snippet shows our logic behind the AC adjustment. After calculating the initial AC adjustment, we check if the difference between the current and desired temperatures is greater than expected. This step is crucial for minimizing the time required to achieve the optimal room temperature. Our comparison takes into account the average temperatures inside and outside the house.

Our environment communicates asynchronously. If a connection problem occurs, the temperature data is queued in the sensor as required. Similarly, if there is a connection issue on the cloud side, the last adjustment AC temperature data is stored and sent once the connection is restored.

We are running the server in a cloud environment, Google Cloud Platform (GCP), by creating a virtual machine (VM) and cloning the repository onto it. We have configured the

firewall rules in GCP to allow client connections to the VM, and found the VM's external IP address to use as the host in the client's configuration file. Both client and server .env files use the same port, and we have set the server host to 0.0.0.0/0 to accept connections from any IP address. Lastly, we have run the server code on the VM and the client code locally, establishing communication between the two for temperature regulation. Our video shows that we have successfully managed to make the fog environment run without any errors. You can check our repository in Github through:

https://github.com/felix-zailskas/fog-computing.git