Fog Computing (Summer Term 2024)
Prof. Dr. D. Bermbach, N. Japke, T. Dockenfuß

Prototype Assignment Documentation - Group 15

Stanislav Kosorin & Leander Kammermeier
Repository Link

System Design

The system is split into three components; Cloud, Edge, and Sensor (see Figure 1). *Sensor1 and Sensor2* are Sensor instances and collect Velocity and Gyroscope data respectively. These are transmitted to the Edge, which forwards the data to the Cloud. In the Cloud Service, a task for the request is added to a queue, which sends a newly calculated position back to the edge once finished.

Edge and Sensor are written in Go, while Cloud is written in Python. The project utilizes Protocol Buffers (Protobuf) to define all interfaces and messages in a single file. These definitions are then compiled into Go and Python representations, ensuring interoperability and providing type safety for each endpoint and message. This approach makes it straightforward to write correct and safe code in both languages. Additionally, each component is defined as a standalone gRPC service, allowing us to get each server running with just a few lines of code.

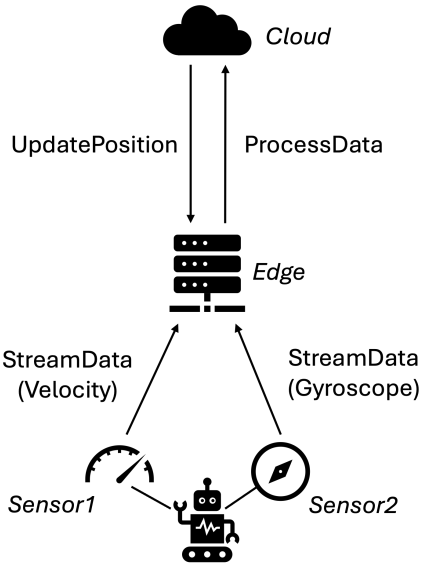

Figure 1: System Design

Sensor 1 & 2

Both sensor components are instances of the same Go server, started with different parameters. Each sensor generates random data of its type (gyroscope or velocity), once every second. This data is available to be consumed from the *StreamData* server stream endpoint (see Figure 2).

```go
func (s *Sensor1Server) StreamData(req *pb.StreamDataRequest, stream pb.SensorService_StreamDataServer) error {
    for {
        data, err := generateData(s.id, s.sensorType)
        if data == nil {
            return fmt.Errorf("Failed to generate data: %v", err)
        }
        if err := stream.Send(&pb.StreamDataResponse{Data: data}); err != nil {
            return err
        }
        time.Sleep(1 * time.Second)
    }
}
```

Figure 2: The StreamData gRPC Server Stream Endpoint

Edge Component

The edge server establishes streams from *Sensor1* and *Sensor2* at startup by sending a request to the *StreamData* endpoint of each sensor. The sensors then start transmitting data continuously. In the event of a node or connection failure, the edge component attempts to reconnect every second in separate goroutines for each sensor (see Figure 3). The same approach is used for cloud connection. This design ensures the edge server is resilient to sensor failures without requiring a restart.

Data from both sensors are queued in a single queue and processed by another goroutine, which forwards the data to the cloud. The queue ensures that data is still collected from the sensors even if the cloud connection is interrupted or the cloud instance crashes. The edge component also exposes the *UpdatePosition* endpoint, which receives positioning data from the cloud, based on the previously transmitted sensor data. This setup simulates a localization algorithm operating on more powerful cloud instances.

```
// Continuously attempt to connect to the sensors
sensor1Addr := net.JoinHostPort(sensor1IP, sensor1Port)
go edgeServer.establishSensorConnection(&edgeServer.sensor1, "1", sensor1Addr)
sensor2Addr := net.JoinHostPort(sensor2IP, sensor2Port)
go edgeServer.establishSensorConnection(&edgeServer.sensor2, "2", sensor2Addr)

// Establish and handle sensor streams
go edgeServer.handleSensorStream(&edgeServer.sensor1, &edgeServer.sensor1Mutex)
go edgeServer.handleSensorStream(&edgeServer.sensor2, &edgeServer.sensor2Mutex)
go edgeServer.processQueue()

// Establish cloud connection
cloudAddr := net.JoinHostPort(cloudIP, cloudPort)
go edgeServer.establishCloudConnection(cloudAddr)
```

Figure 3: Goroutines in EdgeService

Cloud Component

The Cloud component is composed of three classes, *Task*, *TaskQueue,* and *CloudService*.
When the *CloudService* is created by calling the *serve()* method, it is added to a gRPC server where it listens for requests. In addition, a *TaskQueue* is created and filled with the *Tasks* saved in the local replica if there are some present. The *TaskQueue* then processes one *Task* after another, sending their results back to the *Edge Service*, while always staying replicated to the local JSON file. If the *CloudService* receives a new request, a *Task* is created from it and added to the *TaskQueue*, where it will be processed. The *CloudService* returns new position coordinates according to the type of sensor data it has received as seen in Figure 4.

```
def process_task(self, task: Task):
    task_value = float(task.value)
    result_data = {}
    if task.sensor_type == 1:
        result_data = {'x':float(1 * task_value), 'y':float(1 * task_value/2), 'z':float(1 * task_value)}
    else:
        result_data = {'x':float(1 + task_value), 'y':float(1 + task_value/2), 'z':float(1 + task_value)}
    self.task_queue.update_task(task, "result", result_data)
    self.task_queue.update_task(task, "processed", True)

def send_feedback(self, task):
    result_data = task.result
    pos =  fog_pb2.Position(x=result_data.get('x'), y=result_data.get('y'), z=result_data.get('z'))
    return fog_pb2.UpdatePositionResponse(position=pos)
```

Figure 4: Task Processing in CloudService