Step-by-Step Process Machine Failure Prediction with Hadoop MapReduce

1. **Problem Understanding**:
    - The plant uses IoT sensors to monitor machines' temperature, vibration, and pressure.
    - The objective is to **predict potential machine failures** before they happen to **reduce downtime**.
    - We'll analyze historical sensor data to detect anomalies that might indicate impending failure.

2. **Dataset Structure**:
    - Let's assume each sensor reading is recorded with the following format:
        - **Machine ID**: Unique identifier for the machine.
        - **Temperature**: Measured temperature in Celsius.
        - **Vibration**: Measured vibration in vibration units.
        - **Pressure**: Measured pressure in psi.

Create data set as (data.csv) and save it // nano data.csv//

MachineID	Temperature	Vibration	Pressure
Machine1	80.5	45.2	110
Machine2	65.2	30.1	90
Machine3	85.0	60.5	120
Machine4	60.0	40.0	80
Machine5	78.3	52.3	105
Machine6	90.0	48.7	115
Machine7	82.1	51.2	100
Machine8	75.0	41.5	95
Machine9	92.3	63.0	125
Machine10	60.5	38.0	85
Machine11	88.6	47.9	108
Machine12	70.2	44.1	98
Machine13	95.0	55.0	120
Machine14	68.5	42.3	110
Machine15	80.0	49.6	105
Machine16	72.1	36.7	90
Machine17	87.5	58.9	130
Machine18	65.5	39.0	95
Machine19	91.8	64.0	123
Machine20	77.4	50.3	110

Step 1: Upload the Dataset to HDFS:

Before running the MapReduce job, upload your dataset to HDFS:

hdfs dfs -mkdir -p /satwik/predict/input

hdfs dfs -put data.csv /satwik/predict/input

Step 2: Write the MapReduce Code:

In this scenario, we'll focus on detecting potential **machine failure** by analyzing sensor values and identifying **anomalies** in temperature, vibration, or pressure.

//save the file by creating nano filename.java format//

SensorDataMapper.java

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class SensorDataMapper extends Mapper<LongWritable, Text, Text, Text> {
   private static final double TEMP_THRESHOLD = 75.0;
   private static final double VIBRATION_THRESHOLD = 50.0;
   private static final double PRESSURE_THRESHOLD = 100.0;

   @Override
   protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      // Skip the header row
      if (key.get() == 0) {
         return;
      }

      String[] columns = value.toString().split(",");

      // Skip empty lines
      if (columns.length < 4) {
         return;
      }

      try {
         String machineId = columns[0].trim();
         double temperature = Double.parseDouble(columns[1].trim());
         double vibration = Double.parseDouble(columns[2].trim());
         double pressure = Double.parseDouble(columns[3].trim());

         // Check if any threshold is exceeded
         StringBuilder failureReason = new StringBuilder();
         if (temperature > TEMP_THRESHOLD) {
```

```
            failureReason.append("Temperature > 75, ");
        }
        if (vibration > VIBRATION_THRESHOLD) {
            failureReason.append("Vibration > 50, ");
        }
        if (pressure > PRESSURE_THRESHOLD) {
            failureReason.append("Pressure > 100, ");
        }

        // If failure condition exists, output the result
        if (failureReason.length() > 0) {
            failureReason.delete(failureReason.length() - 2, failureReason.length());  // Remove trailing
comma
            context.write(new Text(machineId), new Text("Failure Expected: " +
failureReason.toString()));
        }
      } catch (NumberFormatException e) {
        // Handle invalid numeric data
        System.err.println("Error parsing numeric values: " + value.toString());
      }
   }
}
```

SensorDataReducer.java

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class SensorDataReducer extends Reducer<Text, Text, Text, Text> {

   @Override
   protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
      // Assuming we only have one failure reason per machine
      String failureMessage = "";
      for (Text value : values) {
         failureMessage = value.toString();
      }

      // Output the final failure message for each machine
      context.write(key, new Text(failureMessage));
   }
}
```

MachineFailurePrediction.java

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
```

```
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;


public class MachineFailurePrediction {

   public static void main(String[] args) throws Exception {

      if (args.length != 2) {
         System.err.println("Usage: MachineFailurePrediction <input path> <output path>");
         System.exit(-1);
      }

      // Configure the job
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "Machine Failure Prediction");

      // Set the job's jar file
      job.setJarByClass(MachineFailurePrediction.class);

      // Set Mapper, Reducer classes
      job.setMapperClass(SensorDataMapper.class);
      job.setReducerClass(SensorDataReducer.class);

      // Set output key/value types
      job.setOutputKeyClass(Text.class);
      job.setOutputValueClass(Text.class);

      // Set input and output paths
      FileInputFormat.addInputPath(job, new Path(args[0]));

      Path outputPath = new Path(args[1]);
      FileSystem fs = FileSystem.get(conf);

      // Delete the output directory if it exists
      if (fs.exists(outputPath)) {
         fs.delete(outputPath, true);
      }

      FileOutputFormat.setOutputPath(job, outputPath);

      // Wait for the job to complete
      System.exit(job.waitForCompletion(true) ? 0 : 1);
   }
}
```

Step 3: Compile and Package the Code:

1. **Create the iot_classes directory** :

```
mkdir machine_classes
```

2. **Compile the Java files**:

```
javac -classpath `hadoop classpath` -d machine_classes SensorDataMapper.java
SensorDataReducer.java MachineFailurePrediction.java
```

3. **Package the .class files into a JAR**:

```
jar -cvf MachineFailurePrediction.jar -C machine_classes/ .
```

Step 4: Run the MapReduce Job:

Run the MapReduce job on Hadoop:

```
hadoop jar MachineFailurePrediction.jar MachineFailurePrediction /satwik/predict/input
/satwik/predict/predict_output
```

Step 5: View the Results:

Once the job completes, you can check the output in HDFS:

```
hdfs dfs -cat /satwik/predict/predict_output/part-r-00000
```