实验内容

西安电子科技大学 XIDIAN UNIVERSITY | 计算机科学与技术学院 School of Computer Science and Technology

第3次作业!

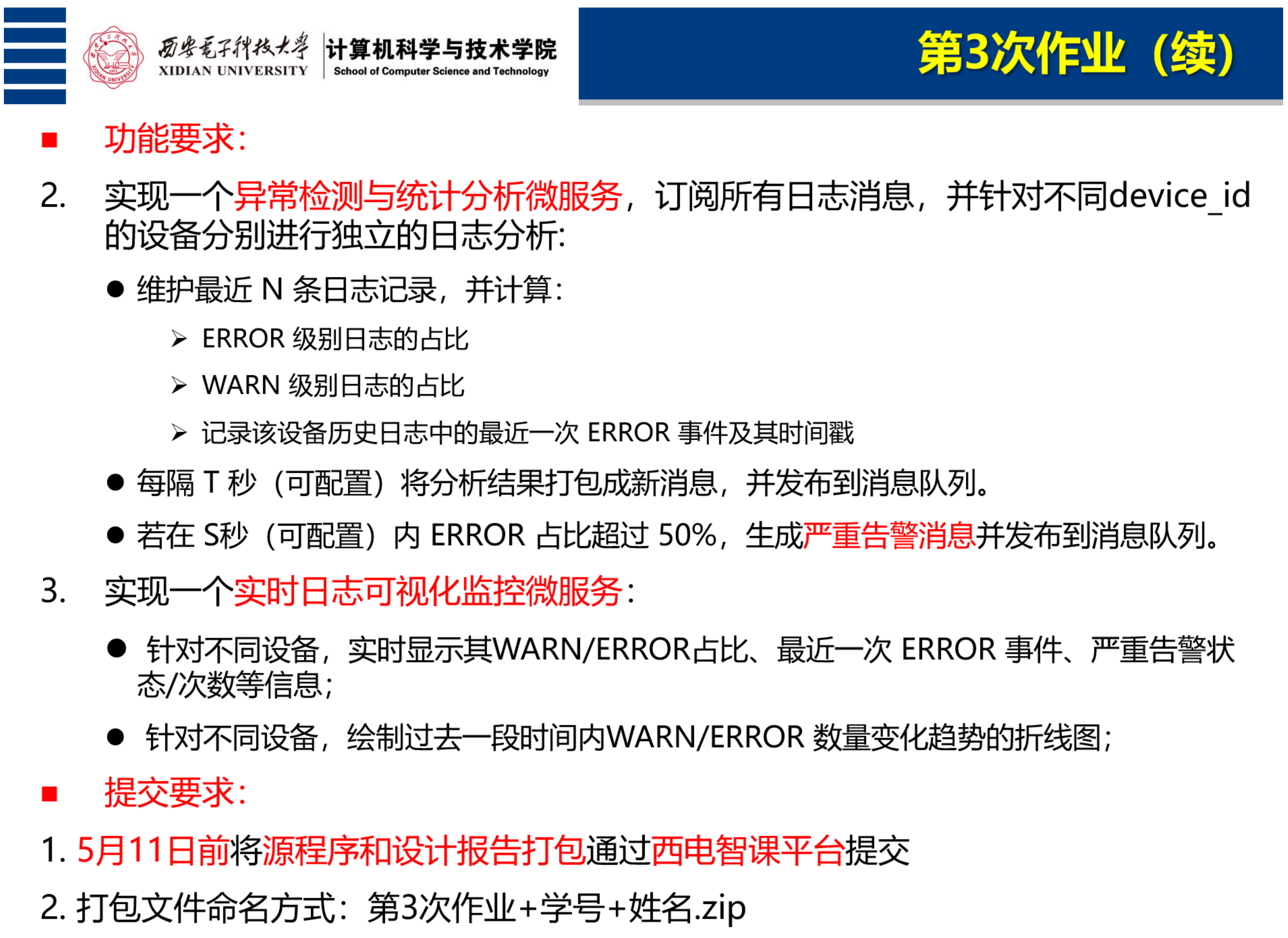
■ 功能要求:

利用 MOM 消息队列技术，实现一个 分布式日志采集分析和异常检测系统，涉及日志采集、异常分析和实时展示。具体要求:

1. 多个日志采集节点:
   - 系统中有多个日志采集节点（实验时用多个进程模拟）， 每个节点用唯一 ID 标识。
   - 每个节点模拟不同设备的日志，每隔 100 毫秒生成一条日志消息，并发布到消息队列。
   - 日志消息格式（类似JSON格式）如下:

```
{
     "device_id":  “device_id1",
     "timestamp": "2025-03-31 12:00:00",
     "log_level": "INFO/WARN/ERROR",
     "message": "系统状态正常"
 }
```

（接下页）


西安电子科技大学 XIDIAN UNIVERSITY
计算机科学与技术学院 School of Computer Science and Technology

第3次作业（续）

■ 功能要求：

2. 实现一个异常检测与统计分析微服务，订阅所有日志消息，并针对不同device_id的设备分别进行独立的日志分析：
   - 维护最近 N 条日志记录，并计算：
     - ERROR 级别日志的占比
     - WARN 级别日志的占比
     - 记录该设备历史日志中的最近一次 ERROR 事件及其时间戳
   - 每隔 T 秒（可配置）将分析结果打包成新消息，并发布到消息队列。
   - 若在 S秒（可配置）内 ERROR 占比超过 50%，生成严重告警消息并发布到消息队列。
3. 实现一个实时日志可视化监控微服务：
   - 针对不同设备，实时显示其WARN/ERROR占比、最近一次 ERROR 事件、严重告警状态/次数等信息；
   - 针对不同设备，绘制过去一段时间内WARN/ERROR 数量变化趋势的折线图；

■ 提交要求：

1. 5月11日前将源程序和设计报告打包通过西电智课平台提交
2. 打包文件命名方式：第3次作业+学号+姓名.zip

准备工作

环境配置

根据老师发的教程安装 activemq 即可。

实验环境

系统：Windows 11

语言：Java

IDE：Intellij IDEA

使用 maven 管理项目。

pom.xml 文件如下：

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>HoshiuZ</groupId>
    <artifactId>Log</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>        <maven.compiler.source>19</maven.compiler.source>
        <maven.compiler.target>19</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>        <dependency>
<groupId>org.apache.activemq</groupId>
            <artifactId>activemq-all</artifactId>
            <version>6.1.6</version>
        </dependency>        <dependency>
<groupId>jakarta.jms</groupId>
            <artifactId>jakarta.jms-api</artifactId>
            <version>3.1.0</version>
        </dependency>        <dependency>
<groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-api</artifactId>
            <version>2.17.2</version>
        </dependency>        <dependency>
<groupId>org.apache.logging.log4j</groupId>
            <artifactId>log4j-core</artifactId>
            <version>2.17.2</version>
        </dependency>        <dependency>
<groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>2.13.0</version>
        </dependency>        <dependency>
<groupId>org.jfree</groupId>
            <artifactId>jfreechart</artifactId>
            <version>1.5.3</version>
        </dependency>    </dependencies>
</project>
```

jackson 用于 JSON 与对象的转化，jfreechart 用于画折线图，activemq 为使用的消息队列。

思路分析

先对实验题目进行分析。

对于第一个日志采集节点，显然其是一个生产者；第二个异常检测与统计分析微服务，其显然是一个消费者，但是对于其功能二三，其也要生成消息发送到消息队列，所以其既是一个消费者也是一个生产者；第三个实时日志可视化监控微服务，其显然是一个消费者。

对于发送 `json` 数据，可以考虑使用 java 中的 Jackson 库，可以比较方便的将数据打包成 `json` 发送到消息队列。

对于日志采集节点的“每个节点用唯一 ID 标识”这一要求，可以考虑对该程序每次传一个参数进去来作为 ID 标识。

需要三个消息队列，一个日志队列 `logs`，一个分析结果队列 `analysisResults` ，一个严重告警队列 `criticalAlerts` 。

多个任务采集节点

只需一个 `producer` 。每隔 100 毫秒向日志队列 `logs` 发送一个日志，其 `log_level` 可以随机生成。

异常检测与统计分析微服务

需要一个 `consumer` 和两个 `producer` 。

一直从日志队列 `logs` 接收日志，对每个设备维护最近的 N 条日志，可以用队列实现，当队列大小大于 N 时就弹出元素。

每隔 T 秒将分析结果打包成新消息，这个可以用 java 中的 `Executors.newSingleThreadScheduledExecutor().scheduleAtFixedRate()` 方法来实现，只需再写一个方法来计算结果并发送。可以动态地维护 N 条日志内错误与警告的数量，然后结果直接与队列大小相除即可。

S 秒内的 ERROR 占比，可以对每个设备再维护一个队列，里面存放 S 秒内的日志，当队头日志与当前日志的时间相差超过了 S 秒，就弹出队头。同理动态维护错误的数量，计算结果大于百分之五十的时候就发送严重告警。

由于用到了多线程，所以 `map` 和 `queue` 要用线程安全的 `ConcurrentHashMap` 和 `ConcurrentLinkedDeque`。

实时日志可视化监控微服务

一个纯消费者，从异常检测与统计分析微服务生产的两个队列接收消息，所以是两个 `consumer`。

对于显示 WARN/ERROR 占比、最近一次 ERROR 事件、严重告警状态次数等信息，直接输出消息队列中获取的信息与监听器中统计的信息即可。

对于绘制折线图，可以使用 Java 的 JFreeChart ，在 maven 的 pom.xml 中添加依赖项后即可使用。

项目结构

日志采集节点为类 `Publisher` 。

三个消息队列的类似于 JSON 的消息为类 `LogEntry, AnalysisResult, CriticalAlert` 。

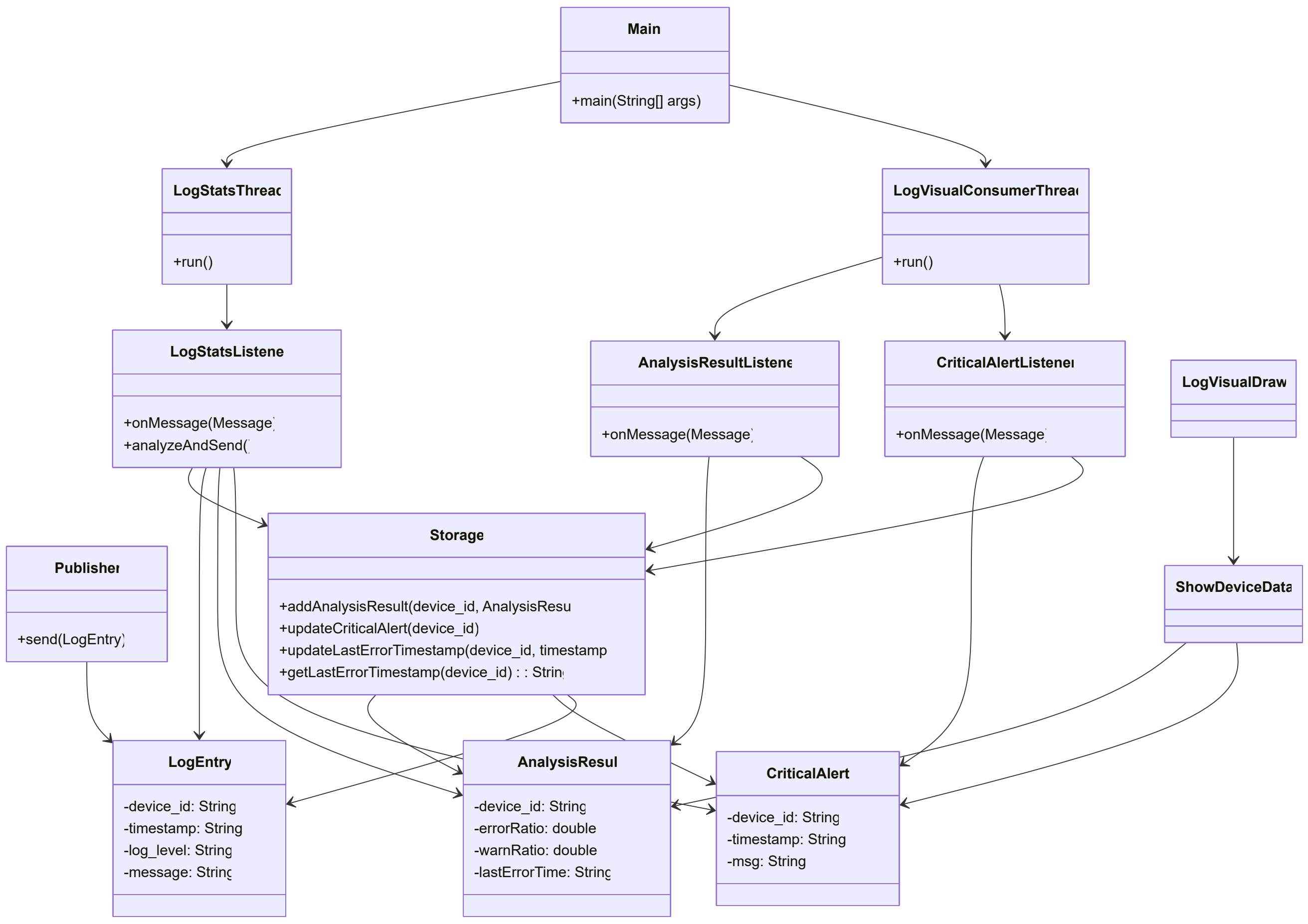
异常检测与统计分析微服务为类 `LogStatsThread` ，其监听器为类 `LogStatsListener`。

实时日志可视化监控微服务的消费者类为类 `LogVisualConsumerThread`，其两个监听器分别为类 `AnalysisResultListener` 与 `CriticalAlertListener`。显示数据为类 `ShowDeviceData`，画图为类 `LogVisualDraw`。

多类共用的数据用了类 `Storage` 来处理。

主类 Main 用于多线程启用功能。

结构图如下：



代码

LogEntry 类

```
public class LogEntry {
    public String device_id;
    public String timestamp;
    public String log_level;
    public String message;

    public LogEntry() {}

    public LogEntry(String device_id, String timestamp, String log_level,
String message) {
        this.device_id = device_id;
        this.timestamp = timestamp;
        this.log_level = log_level;
        this.message = message;
    }
}
```

Publisher 类

```
public class Publisher {
    private static String brokerURL = "tcp://localhost:61616";
    private static ConnectionFactory factory;
    private static Connection connection;
    private Session session;
    private MessageProducer producer;
    private String deviceid;

    public Publisher() throws JMSException {
        factory = new ActiveMQConnectionFactory(brokerURL);
        connection = factory.createConnection();
        connection.start();
        session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        producer = session.createProducer(null);
        deviceid = null;
    }

    public void close() throws JMSException {
        if (connection != null) {
            connection.close();
        }
    }

    public void sendMessage() throws JMSException {
        Destination destination = session.createQueue("logs");
        ObjectMapper mapper = new ObjectMapper();
        Random random = new Random();
        String[] levels = {"INFO", "WARN", "ERROR"};
        while(true) {
            String timestamp =
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"));
            String level = levels[random.nextInt(levels.length)];
            String message = switch (level) {
                case "WARN" -> "警告";
                case "ERROR" -> "错误";
                default -> "正常";
            };

            LogEntry logentry = new LogEntry(deviceid, timestamp, level,
message);
            String json = "";
            try {
                json = mapper.writeValueAsString(logentry);
            } catch (Exception e) {
                e.printStackTrace();
            }

            Message msg = session.createTextMessage(json);
            producer.send(destination, msg);
            System.out.println("Sent message: " + json);

            try{
                Thread.sleep(100);
            } catch(InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }

    public static void main(String[] args) throws JMSException {
        if(args.length == 0) {
            System.out.println("Please provide the device_id as a program
parameter.");
            return;
        }
        Publisher publisher = new Publisher();
        publisher.deviceid = args[0];
        publisher.sendMessage();
        publisher.close();
    }
}
```

AnalysisResult 类

```
public class AnalysisResult {
    public String device_id;
    public double errorRatio;
    public double warnRatio;
    public String lastErrorTime;

    public AnalysisResult() {}

    public AnalysisResult(String device_id, double errorRatio, double
warnRatio, String lastErrorTime) {
        this.device_id = device_id;
        this.errorRatio = errorRatio;
        this.warnRatio = warnRatio;
        this.lastErrorTime = lastErrorTime;
    }
}
```

CriticalAlert 类

```
public class CriticalAlert {
    public String device_id;
    public String timestamp;
    public String msg;

    public CriticalAlert() {}
    public CriticalAlert(String device_id, String timestamp, String msg) {
        this.device_id = device_id;
        this.timestamp = timestamp;
        this.msg = msg;
    }
}
```

Storage 类

```
public class Storage {
    private final Map<String, Deque<AnalysisResult>> analysisResultsQueue =
new ConcurrentHashMap<>();
    private final Map<String, String> lastErrorTimestamp = new
ConcurrentHashMap<>();
    private final Map<String, Boolean> deviceIdFlag = new
ConcurrentHashMap<>();
    private int criticalAlertCount = 0;
    private static final int MAX_HISTORY_SIZE = 20;

    public void addAnalysisResult(String deviceId, AnalysisResult
analysisResult) {
        deviceIdFlag.put(deviceId, true);
        analysisResultsQueue.compute(deviceId, (k, deque) -> {
            if(deque == null) {
                deque = new ConcurrentLinkedDeque<>();
            }
            deque.addLast(analysisResult);
            if(deque.size() > MAX_HISTORY_SIZE) {
                deque.removeFirst();
            }
            return deque;
        });
    }

    public boolean checkDeviceId(String deviceId) {
        return deviceIdFlag.containsKey(deviceId);
    }

    public void updateCriticalAlert(String deviceId, CriticalAlert
criticalAlert) {
        criticalAlertCount++;
    }

    public int getCriticalAlertCount() {
        return criticalAlertCount;
    }

    public Deque<AnalysisResult> getAnalysisResults(String deviceId) {
        return analysisResultsQueue.getOrDefault(deviceId, new
ConcurrentLinkedDeque<>());
    }

    public void updateLastErrorTimestamp(String deviceId, String timestamp)
{
        lastErrorTimestamp.put(deviceId, timestamp);
    }

    public String getLastErrorTimestamp(String deviceId) {
        return lastErrorTimestamp.getOrDefault(deviceId, "No error.");
    }

}
```

LogStatsListener 类

```
public class LogStatsListener implements MessageListener {
    private static final int N = 200;
    private static final int T = 5;
    private static final int S = 1;
    private final Storage storage;

    private Map<String, Deque<LogEntry>> deviceLogs = new
ConcurrentHashMap<>();
    private Map<String, Integer> errorCount = new ConcurrentHashMap<>();
    private Map<String, Integer> warnCount = new ConcurrentHashMap<>();
    private Map<String, Deque<LogEntry>> deviceLogsWithinSSeconds = new
ConcurrentHashMap<>();
    private Map<String, Integer> errorCountWithinSSeconds = new
ConcurrentHashMap<>();
    private Session session;
    private MessageProducer producer1, producer2;

    private static class TimedLogEntry{
        long timestamp;
        String logLevel;

        TimedLogEntry(long timestamp, String logLevel){
            this.timestamp = timestamp;
            this.logLevel = logLevel;
        }
    }

    private static class TimeUtil{
        private static final DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

        public static long toTimestampMillis(String timeStr){
            LocalDateTime localDateTime = LocalDateTime.parse(timeStr,
formatter);
            return
localDateTime.atZone(ZoneId.systemDefault()).toInstant().toEpochMilli();
        }
    }

    public LogStatsListener(Session session, MessageProducer producer1,
MessageProducer producer2, Storage storage) throws JMSException {
        this.session = session;
        this.producer1 =
session.createProducer(session.createQueue("analysisResults"));
        this.producer2 =
session.createProducer(session.createQueue("criticalAlerts"));
        this.storage = storage;


Executors.newSingleThreadScheduledExecutor().scheduleAtFixedRate(this::analy
zeAndSend, T, T, TimeUnit.SECONDS);
    }

    @Override
    public void onMessage(Message message) {
        try{
            if(!(message instanceof TextMessage textMessage)) return;
            String msg = textMessage.getText();

            ObjectMapper mapper = new ObjectMapper();
            LogEntry log = null;
            try {
                log = mapper.readValue(msg, LogEntry.class);
            } catch (JsonProcessingException e) {
                e.printStackTrace();
                return;
            }
            String deviceId = log.device_id;
            deviceLogs.putIfAbsent(deviceId, new ConcurrentLinkedDeque<>());
            Deque<LogEntry> logs = deviceLogs.get(deviceId);

            if(logs.size() >= N) {
                LogEntry removedLog = logs.pollFirst();
                if("ERROR".equalsIgnoreCase(removedLog.log_level)) {
                    errorCount.put(deviceId, errorCount.get(deviceId) - 1);
                } else if("WARN".equalsIgnoreCase(removedLog.log_level)) {
                    warnCount.put(deviceId, warnCount.get(deviceId) - 1);
                }
            }
            logs.addLast(log);

            if("ERROR".equalsIgnoreCase(log.log_level)) {
                errorCount.put(deviceId, errorCount.getOrDefault(deviceId,
0) + 1);
                storage.updateLastErrorTimestamp(deviceId, log.timestamp);
            } else if("WARN".equalsIgnoreCase(log.log_level)) {
                warnCount.put(deviceId, warnCount.getOrDefault(deviceId, 0)
+ 1);
            }

            deviceLogsWithinSSeconds.putIfAbsent(deviceId, new
ConcurrentLinkedDeque<>());
            logs = deviceLogsWithinSSeconds.get(deviceId);
            DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-
MM-dd HH:mm:ss");
            LocalDateTime now = LocalDateTime.now();
            while(!logs.isEmpty()) {
                LogEntry firstLog = logs.peekFirst();
                LocalDateTime firstTime =
LocalDateTime.parse(firstLog.timestamp, formatter);
                if(java.time.Duration.between(firstTime, now).getSeconds() >
S) {
                    LogEntry removedLog = logs.pollFirst();
                    if("ERROR".equalsIgnoreCase(removedLog.log_level)) {
                        errorCountWithinSSeconds.put(deviceId,
errorCountWithinSSeconds.get(deviceId) - 1);
                    }
                } else {
                    break;
                }
            }
            logs.addLast(log);
            if("ERROR".equalsIgnoreCase(log.log_level)) {
                errorCountWithinSSeconds.put(deviceId,
errorCountWithinSSeconds.getOrDefault(deviceId, 0) + 1);
            }
        } catch (JMSException e) {
                e.printStackTrace();
            }
        }

    private void analyzeAndSend() {
        long currentTime = System.currentTimeMillis();
        ObjectMapper mapper = new ObjectMapper();

        for(String deviceId : deviceLogs.keySet()) {
            Deque<LogEntry> logs = deviceLogs.get(deviceId);
            int total = logs.size();
            int errors = errorCount.getOrDefault(deviceId, 0);
            int warns = warnCount.getOrDefault(deviceId, 0);

            double errorRatio = total > 0 ? (double) errors / total : 0.0;
            double warnRatio = total > 0 ? (double) warns / total : 0.0;
            String lastErrorTime = storage.getLastErrorTimestamp(deviceId);

            AnalysisResult analysisResult = new AnalysisResult(deviceId,
errorRatio, warnRatio, lastErrorTime);

            try{
                String analysisJson =
mapper.writeValueAsString(analysisResult);
                Message analysisMsg =
session.createTextMessage(analysisJson);

                producer1.send(analysisMsg);

                logs = deviceLogsWithinSSeconds.get(deviceId);
                int totalWithinSSeconds = logs.size();
                int errorsWithinSseconds =
errorCountWithinSSeconds.get(deviceId);
                double errorRatioWithinSseconds = totalWithinSSeconds > 0 ?
(double) errorsWithinSseconds / totalWithinSSeconds : 0.0;
                if(errorRatioWithinSseconds > 0.5) {
                    String timestamp =
LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss"));
                    String msg = String.format("The ERROR rate over the past
%d seconds has exceeded 50%%, with a ratio of %d/%d", S,
errorsWithinSseconds, totalWithinSSeconds);
                    CriticalAlert criticalAlert = new
CriticalAlert(deviceId, timestamp, msg);
                    String alertJson =
mapper.writeValueAsString(criticalAlert);
                    Message alertMsg = session.createTextMessage(alertJson);
                    producer2.send(alertMsg);
                }
            } catch (JsonProcessingException | JMSException e) {
                e.printStackTrace();
            }
        }
    }
}
```

LogStatsThread 类

```
public class LogStatsThread implements Runnable {
    private static String brokerURL = "tcp://localhost:61616";
    private static ConnectionFactory factory;
    private static Connection connection;
    private Session session;
    private MessageProducer producer1, producer2;
    private MessageConsumer consumer;
    private LogStatsListener logStatsListener;
    private final Storage storage;

    public LogStatsThread(Storage storage) {
        this.storage = storage;
    }

    public void close() throws JMSException {
        if (connection != null) {
            connection.close();
        }
    }

    @Override
    public void run() {
        try {
            factory = new ActiveMQConnectionFactory(brokerURL);
            connection = factory.createConnection();
            connection.start();
            session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
            producer1 =
session.createProducer(session.createQueue("analysisResults"));
            producer2 =
session.createProducer(session.createQueue("criticalAlerts"));
            consumer = session.createConsumer(session.createQueue("logs"));
            logStatsListener = new LogStatsListener(session, producer1,
producer2, storage);

            consumer.setMessageListener(logStatsListener);

            System.in.read();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

AnalysisResultListener 类

```
public class AnalysisResultListener implements MessageListener {
    private final Storage storage;

    public AnalysisResultListener(Storage storage) {
        this.storage = storage;
    }

    @Override
    public void onMessage(Message message) {
        try{
            if(!(message instanceof TextMessage textMessage)) return;
            String msg = textMessage.getText();

            ObjectMapper mapper = new ObjectMapper();
            AnalysisResult analysisResult = mapper.readValue(msg,
AnalysisResult.class);

            storage.addAnalysisResult(analysisResult.device_id,
analysisResult);
        } catch (JMSException | JsonProcessingException e) {
            e.printStackTrace();
        }
    }
}
```

CriticalAlertListener 类

```
public class CriticalAlertListener implements MessageListener {
    private final Storage storage;

    public CriticalAlertListener(Storage storage) {
        this.storage = storage;
    }

    @Override
    public void onMessage(Message message) {
        try{
            if(!(message instanceof TextMessage textMessage)) return;
            String msg = textMessage.getText();
            ObjectMapper mapper = new ObjectMapper();
            CriticalAlert criticalAlert = mapper.readValue(msg,
CriticalAlert.class);

            storage.updateCriticalAlert(criticalAlert.device_id,
criticalAlert);

        } catch (JMSException | JsonProcessingException e) {
            e.printStackTrace();
        }
    }
}
```

ShowDeviceData 类

```
public class ShowDeviceData {
    private final Storage storage;

    public ShowDeviceData(Storage storage) {
        this.storage = storage;
    }

    public void showData(String deviceId) {
        Deque<AnalysisResult> analysisResults =
storage.getAnalysisResults(deviceId);
        if(!analysisResults.isEmpty()) {
            AnalysisResult latestResult = analysisResults.peekLast();
            System.out.printf("Now the ratio of error is %.2f%%.%n",
latestResult.errorRatio * 100);
            System.out.printf("Now the ratio of warn is %.2f%%.%n",
latestResult.warnRatio * 100);
            System.out.printf("The latest error timestamp is %s.%n",
storage.getLastErrorTimestamp(deviceId));
        }
    }
}
```

LogVisualDraw 类

```
public class LogVisualDraw {
    private static final int T = 5;

    public static void draw(String deviceId, Deque<AnalysisResult> results)
throws Exception {
        DefaultCategoryDataset dataset = new DefaultCategoryDataset();
        int size = results.size();
        for(AnalysisResult result : results) {
            dataset.addValue(result.errorRatio * 100, "ERROR Ratio",
String.valueOf(-T * size));
            dataset.addValue(result.warnRatio * 100, "WARN Ratio",
String.valueOf(-T * size));
            size--;
        }

        JFreeChart chart = ChartFactory.createLineChart(
                "LogVisualization of device " + deviceId,
                "Time(s)",
                "Ratio(%)",
                dataset,
                PlotOrientation.VERTICAL,
                true,
                true,
                false
        );

        File dir = new File("pic");
        if(!dir.exists()) {
            dir.mkdirs();
        }

        String timestamp = String.valueOf(System.currentTimeMillis() /
1000);
        String filePath = "pic/" + deviceId + "_" + timestamp + ".png";
        File outputFile = new File(filePath);

        ChartUtils.saveChartAsPNG(outputFile, chart, 800, 600);
        System.out.println("The chart is saved to :" +
outputFile.getAbsolutePath());
    }
}
```

LogVisualConsumerThread 类

```
public class LogVisualConsumerThread implements Runnable{
    private final Storage storage;
    private volatile boolean running = true;

    public LogVisualConsumerThread(Storage storage) {
        this.storage = storage;
    }

    @Override
    public void run() {
        try{
            String brokerURL = "tcp://localhost:61616";
            ConnectionFactory factory = new
ActiveMQConnectionFactory(brokerURL);
            Connection connection = factory.createConnection();
            connection.start();
            Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
            MessageConsumer consumer1 =
session.createConsumer(session.createQueue("analysisResults"));
            MessageConsumer consumer2 =
session.createConsumer(session.createQueue("criticalAlerts"));

            AnalysisResultListener analysisResultListener = new
AnalysisResultListener(storage);
            CriticalAlertListener criticalAlertListener = new
CriticalAlertListener(storage);
            consumer1.setMessageListener(analysisResultListener);
            consumer2.setMessageListener(criticalAlertListener);

            System.in.read();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void stop(){
        running = false;
    }
}
```

Main 类

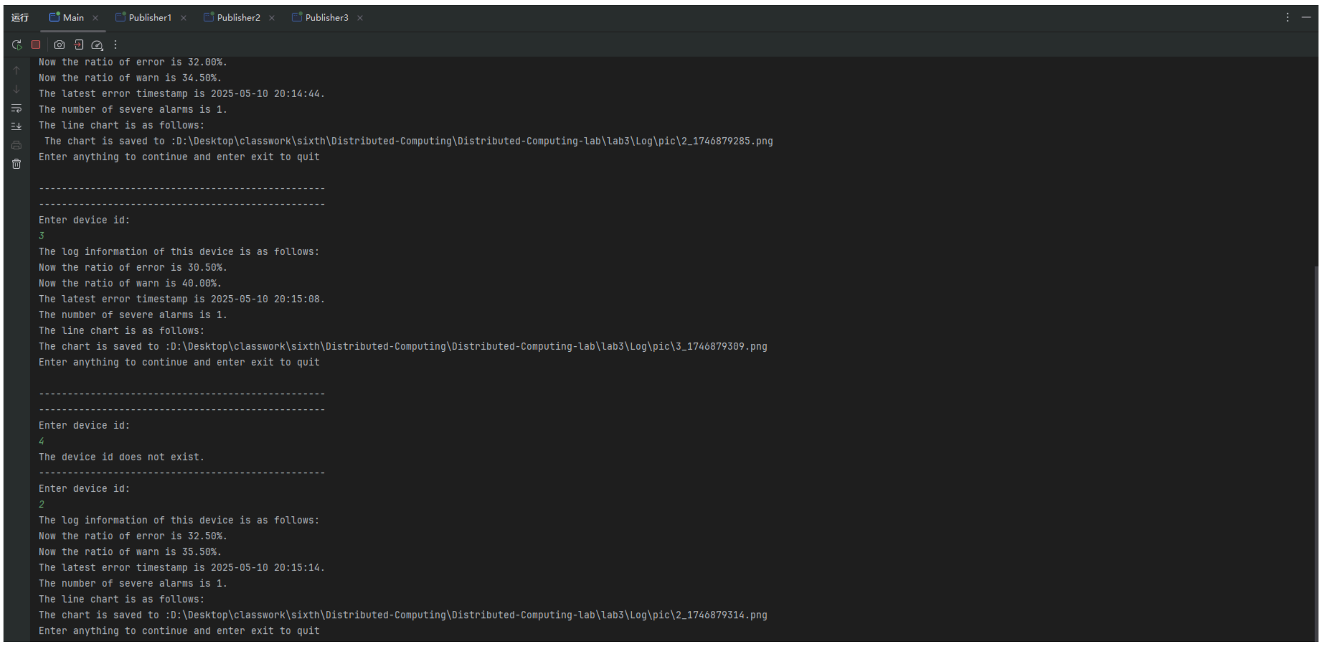
```
public class Main {
    public static void main(String[] args) throws Exception {
        Storage storage = new Storage();
        Thread logStatsThread = new Thread(new LogStatsThread(storage));
        Thread logVisualConsumerThread = new Thread(new
LogVisualConsumerThread(storage));
        ShowDeviceData showDeviceData = new ShowDeviceData(storage);

        logStatsThread.start();
        logVisualConsumerThread.start();

        String choice = "1";
        while(!"exit".equalsIgnoreCase(choice)) {
            Scanner sc = new Scanner(System.in);
            System.out.println("---------------------------------------------
-------");
            System.out.println("Enter device id: ");
            String deviceId = sc.nextLine();
            if(storage.checkDeviceId(deviceId)) {
                System.out.println("The log information of this device is as
follows: ");
                showDeviceData.showData(deviceId);
                System.out.println("The number of severe alarms is " +
storage.getCriticalAlertCount() + ".");
                System.out.println("The line chart is as follows: ");
                Deque<AnalysisResult> result =
storage.getAnalysisResults(deviceId);
                LogVisualDraw.draw(deviceId, result);
                System.out.println("Enter anything to continue and enter
exit to quit");
                choice = sc.nextLine();
                if ("exit".equalsIgnoreCase(choice)) {
                    break;
                }
                System.out.println("-----------------------------------------
----------");
            }
            else {
                System.out.println("The device id does not exist.");
            }
        }
    }
}
```

运行示例

开了三个 publisher，deviceid 分别为 1, 2, 3.

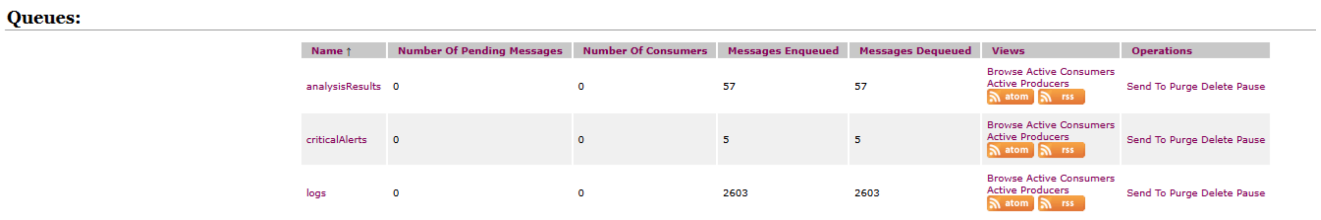


```
Now the ratio of error is 32.00%.
Now the ratio of warn is 34.50%.
The latest error timestamp is 2025-05-10 20:14:44.
The number of severe alarms is 1.
The line chart is as follows:
 The chart is saved to :D:\Desktop\classwork\sixth\Distributed-Computing\Distributed-Computing-lab\lab3\Log\pic\2_1746879285.png
Enter anything to continue and enter exit to quit

--------------------------------------------------
--------------------------------------------------
Enter device id:
3
The log information of this device is as follows:
Now the ratio of error is 30.50%.
Now the ratio of warn is 40.00%.
The latest error timestamp is 2025-05-10 20:15:08.
The number of severe alarms is 1.
The line chart is as follows:
The chart is saved to :D:\Desktop\classwork\sixth\Distributed-Computing\Distributed-Computing-lab\lab3\Log\pic\3_1746879309.png
Enter anything to continue and enter exit to quit

--------------------------------------------------
--------------------------------------------------
Enter device id:
4
The device id does not exist.
--------------------------------------------------
Enter device id:
2
The log information of this device is as follows:
Now the ratio of error is 32.50%.
Now the ratio of warn is 35.50%.
The latest error timestamp is 2025-05-10 20:15:14.
The number of severe alarms is 1.
The line chart is as follows:
The chart is saved to :D:\Desktop\classwork\sixth\Distributed-Computing\Distributed-Computing-lab\lab3\Log\pic\2_1746879314.png
Enter anything to continue and enter exit to quit
```

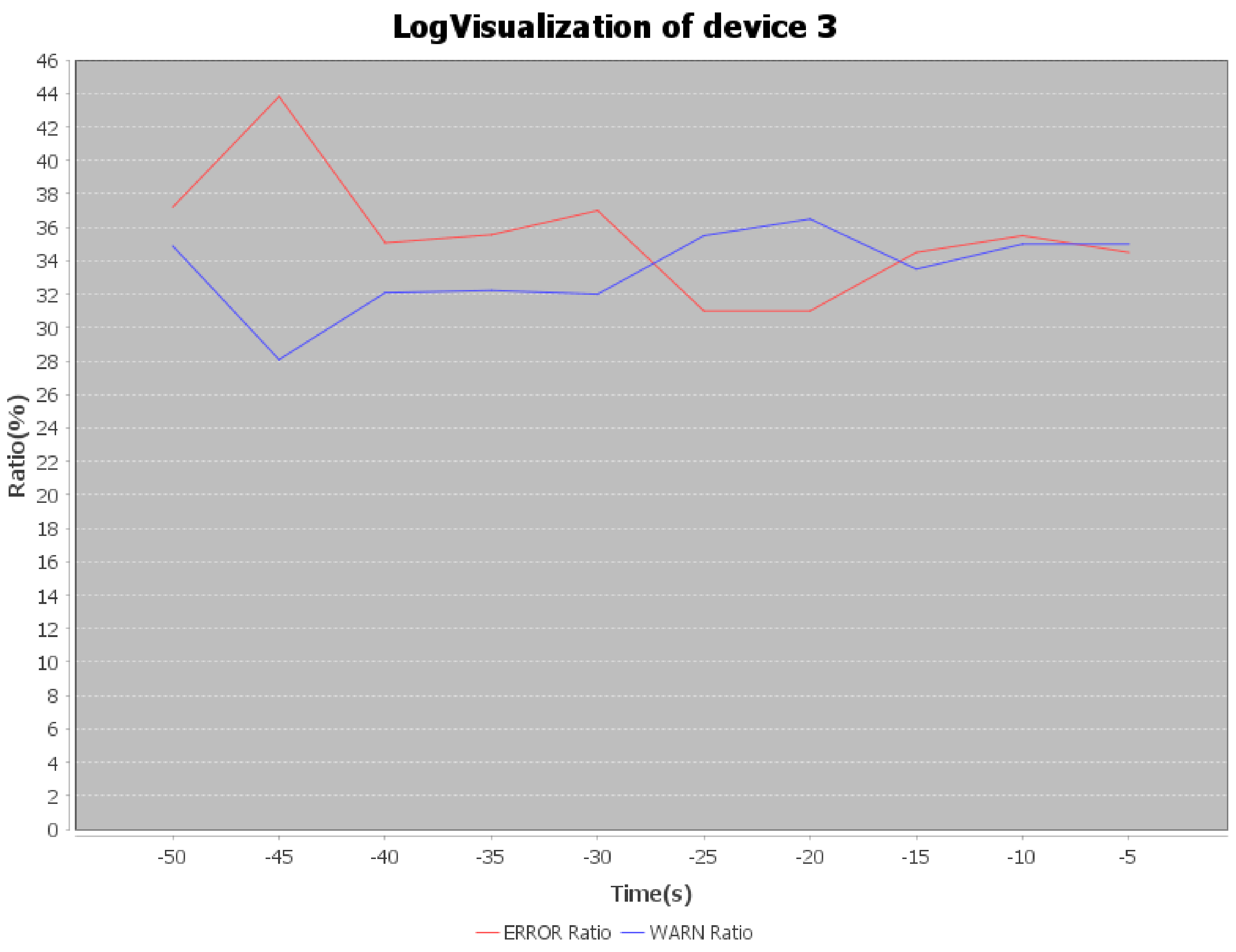
activemq 控制台界面如下：

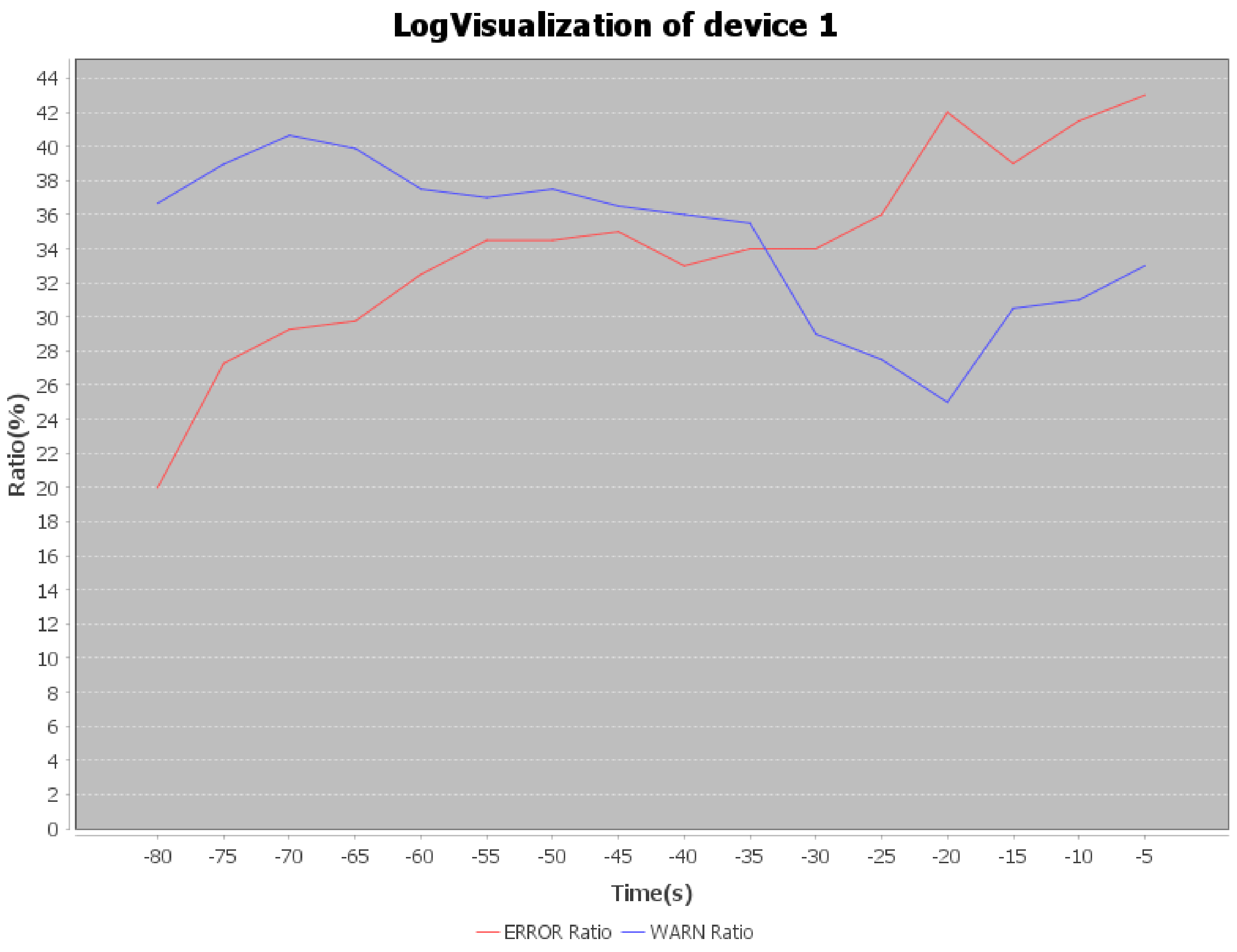


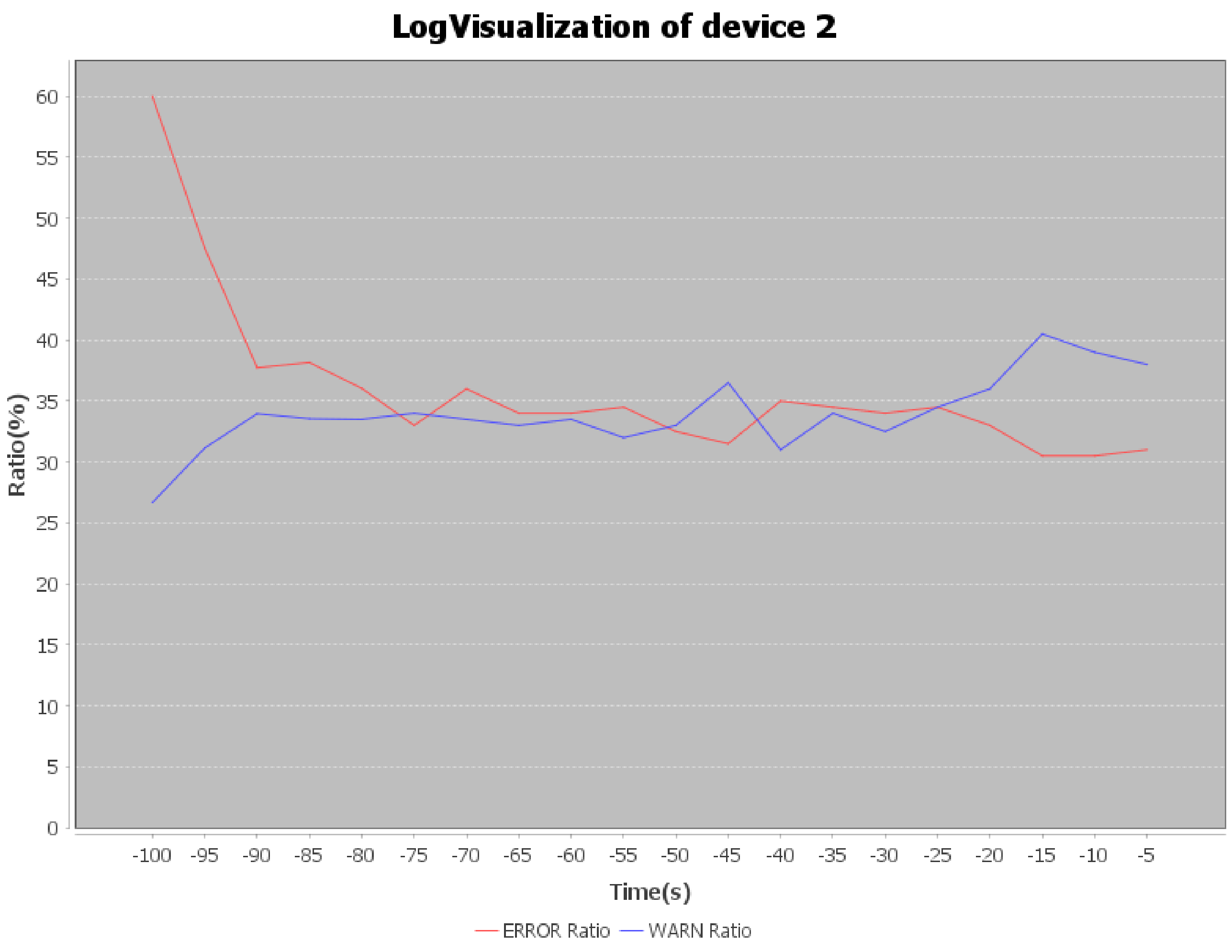
Queues:

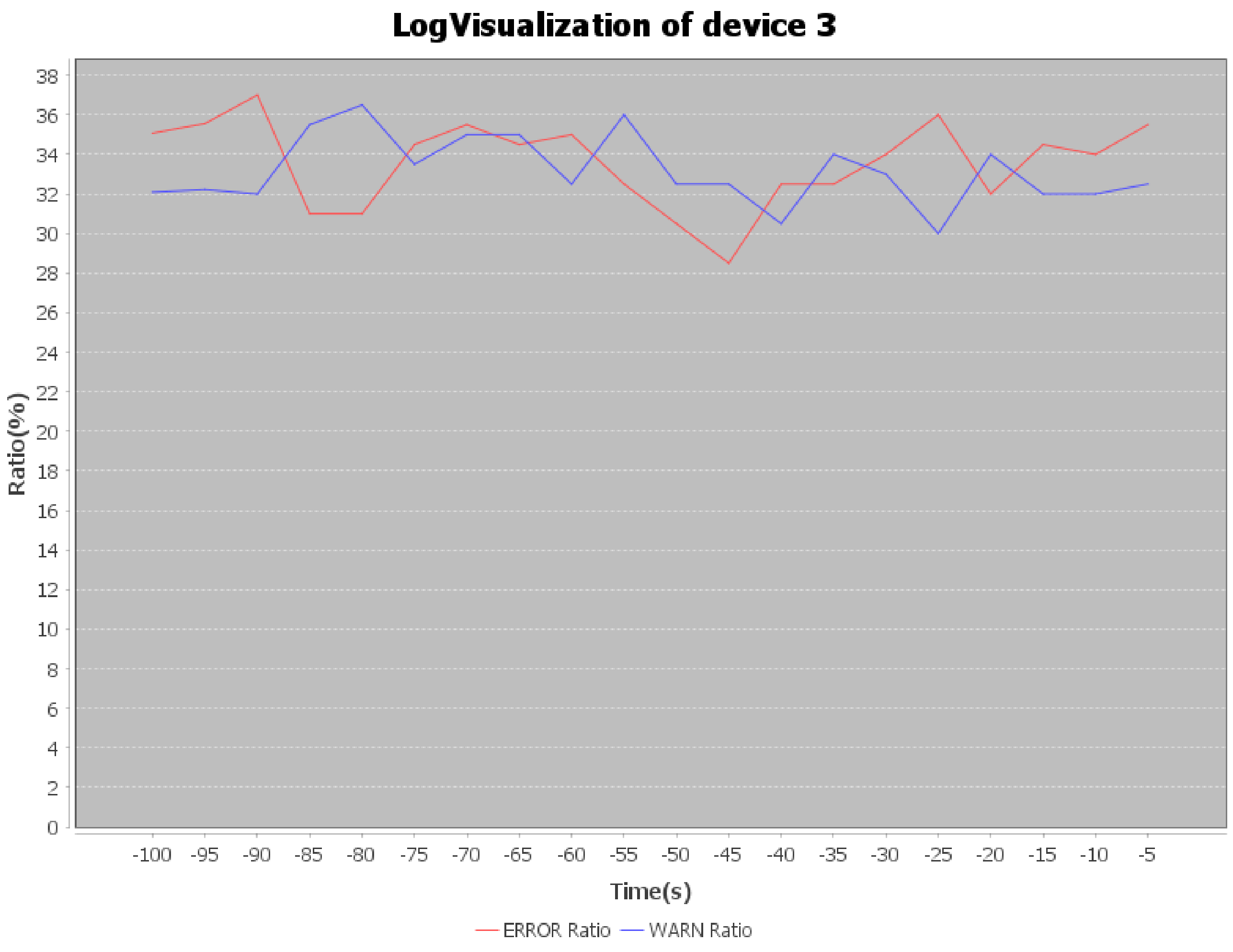
Name ↑	Number Of Pending Messages	Number Of Consumers	Messages Enqueued	Messages Dequeued	Views	Operations
analysisResults	0	0	57	57	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause
criticalAlerts	0	0	5	5	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause
logs	0	0	2603	2603	Browse Active Consumers Active Producers atom rss	Send To Purge Delete Pause

得到的部分折线图如下：


LogVisualization of device 3
Ratio(%)
Time(s)
ERROR Ratio
WARN Ratio


LogVisualization of device 1
Ratio(%)
Time(s)
ERROR Ratio
WARN Ratio


LogVisualization of device 2
Ratio(%)
0
5
10
15
20
25
30
35
40
45
50
55
60
-100
-95
-90
-85
-80
-75
-70
-65
-60
-55
-50
-45
-40
-35
-30
-25
-20
-15
-10
-5
Time(s)
ERROR Ratio
WARN Ratio


LogVisualization of device 3
Ratio(%)
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
-100
-95
-90
-85
-80
-75
-70
-65
-60
-55
-50
-45
-40
-35
-30
-25
-20
-15
-10
-5
Time(s)
ERROR Ratio
WARN Ratio