Distributed Algo – Exercise 3 (Group 06)

Darshan Hingu 380584
RaviPrasad Marike Ramesh 387219
Seema Narasimha Swamy 384418
Yuchun Chen 387275

Exercise 3.1: Vector Clocks

i) Causal Order:

The concept of the "causal broadcasat" has been introduced in the lecture as an application of vector clocks. In order to guarantee the proper execution of the algorithm it was necessary to send each message to all participating nodes. If we want to avoid sending each message to all nodes, why wouldn't it be sufficient to only use the vector as applied by the causal broadcast to achieve causal order?

If any process pi delivers a message m2, then pi must have delivered every message m1 such that m1 -> m2. If not sending each message to all nodes and there is insufficient message been generated.

Since it allow ensure causal order at receiver, but it is not possible to detect if messages with smaller time stamps are missing.

ii) Order Relation :

If e is a system event, then V (e) defines its vector time stamp and C(e)$\in$N is the corresponding sum of the vector's components. Whenever an event e occurs, V (e) is computed as introduced in the lecture.

1. Show that C(e) together with the "less than"relation (<) defines a partial

   order of the set of system events.

   With above condition along with C(e), we can not distinguish the concurrent events

   for example:V(A)=(1,0,0)=>c(A)=1 AND V(B)=(0,1,1)=>c(B)=2

   AND A|| B BUT c(A)<c(B) !A->B.

   By adding some condtion, like a->b, then we are able to comeout the corresponding C of both events as below :

   V(A)=(1,0,0)=>c(A)=1 AND V(B)=(1,0,1)=>c(B)=2 then c(A)<c(B) .

2. Show that the respective logical clock fulfills the clock condition.
3. Show that the partial order can be extended to a total order by applying

process IDs together with the vector time stamp.

If the application requires total order this could be enforced by modifying the vector clock algorithm to include ACKs.

p1 sent a message with timestamp 1.

All delivery systems collect the message, multicast ACK and collect all ACKs if there is no other messages with same timestamp receive the message and change their timestamp. Otherwise , if there is a contention(two message which both condition T[i] = Vj[i] + 1∧∀ k≠i:T[k] ≤ Vj [k] is true for them) - use a tie-breaker (e.g. Lowest process ID wins) and deliver the message from smaller process id first.

Exercise 3.2: [(DiveSurf Inc.) Integration Scenario]

Entire Src Code and JAR file you can find in the below link.

https://gitlab.tubit.tu-berlin.de/hingudarshan/dalgo_ex3

a) WebOrderSystem:

```
public class WebOrderSystem {
  private static int orderID = 1;
      /* Converts the incoming new order to the format expected by Billing and Inventory System*/
      private static Processor orderProcessor = new Processor() {
          @Override
          public void process(Exchange exchange) throws Exception {
              String[] parts = exchange.getIn().getBody(String.class).split(",");
              String FirstName = parts[0];
              String LastName = parts[1];
              int numOrderedSurfBoards = Integer.parseInt(parts[2]);
              int numOrderedDivingSuits = Integer.parseInt(parts[3]);
              int customerId = Integer.parseInt(parts[4]);

              System.out.println("CamelMain: process(): customerId: " + customerId);
              /* Create a transformedOrder object and set as body of exchange*/
              exchange.getIn().setBody(new TransformedOrder(customerId, FirstName, LastName,
                      numOrderedDivingSuits + numOrderedSurfBoards,
                      numOrderedDivingSuits,numOrderedSurfBoards,orderID, false));
              orderID++;
          }
      };
  public static void main(String[] args) throws Exception {
      DefaultCamelContext ctxt = new DefaultCamelContext();
      ActiveMQComponent activeMQComponent = ActiveMQComponent.activeMQComponent();
      activeMQComponent.setTrustAllPackages(true);
      ctxt.addComponent("activemq", activeMQComponent);
      RouteBuilder route = new RouteBuilder() {
          @Override
          public void configure() throws Exception {
              from("file:/Users/darshan/Downloads/Task3/src/main/java/com/tub/webOrders?noop=true")
                      .split(body().tokenize("\n"))
                      //convert to string class as expected by CallcenterOrderSystem
                      .convertBodyTo(String.class)
                      //add to callCenter activemq - Channel adapter endpoint
```

```
                    .to("activemq:queue:orderQueue")
                    .process(orderProcessor)
                    //using pub-sub pattern , send from weborder to billing,inventory
                    .to("activemq:queue:billing", "activemq:queue:inventory");

        }
    };

    ctxt.addRoutes(route);
    ctxt.start();
    System.in.read();
    ctxt.stop();
  }
}
```

b) CallCenterOrderSystem

```
public class CallCenterOrderSystem {
    //Convert the incoming order to the required format to write to a file(Message transformer/Cont
    private static Processor callCenterOrderProcessor = new Processor() {
        @Override
        public void process(Exchange exchange) throws Exception {
            String[] parts = exchange.getIn().getBody(String.class).split(",");
            String newOrderFormat;
            if (parts.length == 5) {
                String firstName = parts[0];
                String lastName = parts[1];
                String numSurfBoards = parts[2];
                String numDivingSuits = parts[3];
                String customerId = parts[4];
                newOrderFormat = customerId + "," + firstName + " " + lastName + "," + numSurfBoard
                        + "," + numDivingSuits + "\n";
            }
            else
            {
                newOrderFormat = exchange.getIn().getBody(String.class);
            }
            exchange.getIn().setBody(newOrderFormat);
        }
    };

  /*Aggregator used to aggregate messages and print every 2 minutes into file*/
  public static class CountingAggregation implements AggregationStrategy {

      @Override
      public Exchange aggregate(Exchange exchange, Exchange exchange1) {
          if (exchange == null) {
              return exchange1;
          }

          Object body = exchange.getIn().getBody();
          Object body1 = exchange1.getIn().getBody();
          exchange1.getIn().setBody(body + "" + body1);

          return exchange1;
      }
  }

  public static void main(String[] args) throws Exception {
      DefaultCamelContext ctxt = new DefaultCamelContext();
      ActiveMQComponent activeMQComponent = ActiveMQComponent.activeMQComponent();
      activeMQComponent.setTrustAllPackages(true);
      ctxt.addComponent("activemq", activeMQComponent);

      RouteBuilder route = new RouteBuilder() {
```

```
        @Override
        public void configure() throws Exception {
            //read from the queue
            from("activemq:queue:orderQueue")
                    .process(callCenterOrderProcessor)
                    //aggregate messages ,wait for 2 minutes//reduce to 10 sec
                    .aggregate(constant(0), new CallCenterOrderSystem.CountingAggregation()).comple
                    //write to a file
                    .to("file:/Users/darshan/Downloads/Task3/src/main/java/com/tub/callCenterOrders
                    //Testcode, output to stream to check the interval of aggregation
                    .to("stream:out");
        }
    };
    ctxt.addRoutes(route);
    ctxt.start();
    System.in.read();
    ctxt.stop();
  }
}
```

c) BillingSystem

```
public class BillingSystem {
    //Processor which handles the task of credit check for the customer
    private static Processor BillingProcessor = new Processor() {
        @Override
        public void process(Exchange exchange) throws Exception {
            TransformedOrder order = (TransformedOrder) exchange.getIn().getBody();
            //validation is done using a random function as we dont have customer data
            boolean validated = Math.random() > 0.5;
            order.setValid(validated);

            System.out.println(order);
            //header set for aggregation
            exchange.getIn().setHeader("OrderId", order.getOrderID());
            exchange.getIn().setBody(order);
        }
    };

  public static void main(String[] args) throws Exception{
    DefaultCamelContext ctxt = new DefaultCamelContext();
    ActiveMQComponent activeMQComponent = ActiveMQComponent.activeMQComponent();
    activeMQComponent.setTrustAllPackages(true);
    ctxt.addComponent("activemq", activeMQComponent);

    RouteBuilder route = new RouteBuilder() {
        @Override
        public void configure() throws Exception {
            //read from pub-sub channel and write to aggregator queue
            from("activemq:queue:billing")
                    .process(BillingProcessor)
                    //point to point channel for aggregation and result system
                    .to("activemq:queue:aggregatorQueue");
        }
    };

    ctxt.addRoutes(route);

    ctxt.start();
    System.in.read();
    ctxt.stop();
  }
}
```

d) InventorySystem:

```
public class InventorySystem {
      //Check the availability of the requested items
      private static Processor InventoryProcessor = new Processor() {
          @Override
          public void process(Exchange exchange) throws Exception {
              TransformedOrder order = (TransformedOrder) exchange.getIn().getBody();
              boolean validated = Math.random() > 0.5;
              order.setValid(validated);

              System.out.println(order);
              exchange.getIn().setHeader("OrderId", order.getOrderID());
              exchange.getIn().setBody(order);
          }
      };

  public static void main(String[] args) throws Exception{
      DefaultCamelContext ctxt = new DefaultCamelContext();
      ActiveMQComponent activeMQComponent = ActiveMQComponent.activeMQComponent();
      activeMQComponent.setTrustAllPackages(true);
      ctxt.addComponent("activemq", activeMQComponent);

      RouteBuilder route = new RouteBuilder() {
          @Override
          public void configure() throws Exception {
              //read from pub-sub channel
              from("activemq:queue:inventory")
                      .process(InventoryProcessor)
                      //write to point to point channel for aggregation
                      .to("activemq:queue:aggregatorQueue");
          }
      };

      ctxt.addRoutes(route);
      ctxt.start();
      System.in.read();
      ctxt.stop();
  }
}
```

e) ResultSystem

```
public class ResultsSystem {
  public static void main (String[] args) throws Exception
  {
      DefaultCamelContext ctxt = new DefaultCamelContext();
      ActiveMQComponent activeMQComponent = ActiveMQComponent.activeMQComponent();
      activeMQComponent.setTrustAllPackages(true); //Cannot access jms Component
      ctxt.addComponent("activemq", activeMQComponent);

      RouteBuilder route = new RouteBuilder() {
          @Override
          public void configure() throws Exception {

              from("activemq:queue:aggregatorQueue")
                      //aggregator waits for 2 msgs and send to Result Aggregator
                      .aggregate(header("OrderID"), new
ResultAggregator()).completionSize(2)
                      .choice()
                          //based on aggregation result, if its a valid msg, stream:out
                          .when(header("valid"))
```

```
                            .to("stream:out")
                        //else stream:err
                        .otherwise()
                            .to("stream:err");

            }
        };

        ctxt.addRoutes(route);

        ctxt.start();
        System.in.read();
        ctxt.stop();
    }
}
```