Project #6

Chow Sheung Him Martin, Mitchell Dang, Xinyu Wang, Krish Kalai
Group: 7

Amazon Availability Messaging

Business Problem

Originally, every 15 minutes we run a batch job that figures out the availability for each ASIN. However, if the services get too many orders within 15 minutes, there's no way to keep track of the availability on the website.

Provide a modified schema for a single ASIN on the server (Basically adding Sequence number, Reference count, dirty flag etc as you think is needed to make the system work)

```
{
        ASIN: <string>,
        NumAvail:<integer>,
        Reserved: [{OrderID:<string>, Needs:<integer>}, . . . ]
        FC[ {Name:<String>, InStock:<integer> , FCSeqNum:<integer>} . . .],
        SeqNum:<integer>
}
```

FCSeqNum: this sequence number is updated when the Availability Service receives a new message that has the larger (newer) FCSeqNum from the FC.

SeqNum: This sequence number is updated when a message is sent to the obidos.

Provide a list of relevant events, and the subsequent messages needed to main inventory info, and updating the Availability on the website.
Examples of events - Order Created, Inventory Received in an FC, Shipment Created
Note that you may need additional 'events' (like Midnight occurring) that triggers Messages.

1. OrderCreated
2. OrderCanceled
3. InventoryChanged
4. ShipmentCreate

5. OrderUpdate
6. AvailabilityUpdate

For each event, provide the message(s) that will be produced that will keep the availability appropriately up to date and notify the online boxes of changes to availability.

1. **OrderCreated**
Description: This event triggers after a user creates an order. One message is sent per order identified by *OrderID*. The value *Needs* is the new quantity after the order has been created. After receiving the message, the Availability Service will create the order in Reserved and update the item information in its database.
Publisher: OrderProcessing
Subscriber: Availability

```
OrderCreated: {
        OrderID: <string>,
        Items: [{
                ASIN: <string>,
                Needs: <integer>}, ...]
}
```

2. **OrderCanceled**
Description: This event triggers after a user canceled an order (that has been previously created). One message is sent per canceled order. After receiving the message, the Availability Service will remove that order entry having the matching orderID.
Publisher: OrderProcessing
Subscriber: Availability

```
OrderCanceled: {
        OrderID: <string>
}
```

3. **InventoryChanged**
Description: This event is triggered when the Fulfillment Center has a change in inventory that is not by shipping an item (i.e. receive an item or item found damaged). After receiving the message, the Availability service will check with the FCSeqNum, if the FCSeqNum of the message it received is higher than the current FCSeqNum, it will update the InStock of item information accordingly. Otherwise, it will ignore that message (older message).
Publisher: Fulfillment Center
Subscriber: Availability

```
InventoryChanged: {
        FC: {
```

```
            Name:<String>,
            item: [{ASIN :<String>, InStock:<integer>}, ...],
            FCSeqNum:<integer>
        }
}
```

4. **ShipmentCreate**
Description: This event is triggered by the Fulfillment Center when it ships an order (updating the inventory value). After receiving the message, the Availability service will check with the FCSeqNum, if the FCSeqNum of the message it received is higher than the current FCSeqNum, it will change the InStock of the correlated items in the Fulfillment Center. Otherwise, it will ignore that message (older message). It also removes the order entry in the Availability service.

Publisher: Fulfillment Center
Subscriber: Availability

```
ShipmentCreated: {
        FC: {Name:<String>, item: [{ASIN :<String>, InStock:<integer>}, ...],
        SeqNum:<integer>},
        OrderID: <string>
}
```

5. **OrderUpdate**
Description: The event is triggered once per day at midnight to update the orders of all availability instances. Overwrite the order in Order Processing to Availability.
Publisher: Order Processing
Subscriber: Availability

```
OrderingUpdate: {
        OrderID: <string>,
        item [{ASIN: <string>, number:<integer>}, …]
}
```

6. **AvailabilityUpdate**
Description: The event is triggered once NumAvail is updated in Availability Service. It updates the available number in Obidos if the message has a higher seqNum than the current seqNum.
Publisher: Availability
Subscriber: Obidos

```
AvailabilityUpdate: {
        ASIN:<string>
        NumAvail:<integer>
        SeqNum:<integer>
}
```

Extra Credit (up to 10 points)

To assure availability, you have three services running at all times, all receiving update messages. How do you decide which service should send availability messages to the website? IE how do you know then the leader has died, and how do you decide on the new leader?

To decide which service should send availability messages to the website, we want to find the most recent change of data across the active Databases. In order to do that, one service(every server that gets the request) will send their sum of FCseqNum(s) of the ordered item or change of instock number to other services. If one service receives a sum of FCseqNum(s) that is greater than its sum of FCseqNum(s), it does nothing. If no sum of FCseqNum(s) is received, that service will automatically become the leader (who is responsible to send out the message to Obidos). Under these two circumstances: the other two services have lower sum of FCseqNum(s) or one or two of the other two services are down, then the server that doesn't receive any feedback (higher sum of FCseqNum(s)) is the leader (who is responsible to send out the message to Obidos). If the leader message is lost, more than one of the leaders may send the message. The Obidos will have the correct result because of the sum of FCseqNum(we pass the FCSeqNumsum). If all services have the same sum of FCSeqNum(s), the alive server having a smaller index(ie. server 1) will always be the leader. Otherwise, if that server is death, the next alive server will be picked to be the leader (ie. server 2).

A snippet of a message sent would change to:
Publisher: one Availability
Subscriber: other Availability
AvailabilityData: {
    ASIN: <string>
    FCSeqNumSum : <integer>,
    numAvailable: <integer>,
    SeqNum: <integer>
}