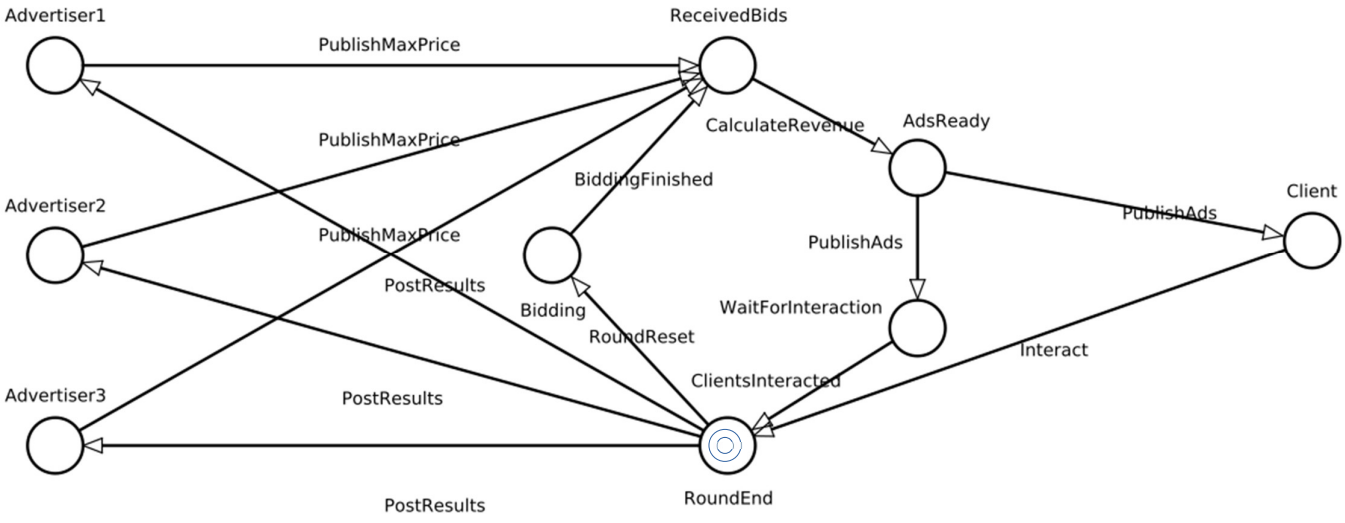
Assignment 3 (mini project)

Christos Papastamos csd4569

Phase A

The Petri net I created for the use case is the following:

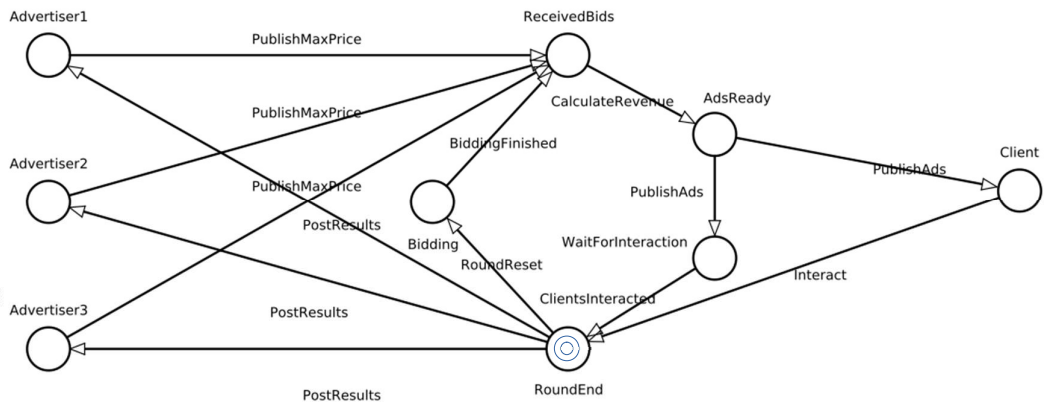


In the above Petri net there can be seen 3 Advertisers (on the left) 1 Client (on the right) and an Ad Service (the five states in the middle).The reachability graph for a vector with definition: (RoundEnd,Advertiser1,Advertiser2,Advertiser3,Bidding,ReceivedBids,AdsReady,Client,WaitForInteraction) would be:

(1,0,0,0,0,0,0,0,0)→(0,1,1,1,1,0,0,0,0)→(0,0,0,0,0,1,0,0,0)→
(0,0,0,0,0,0,0,,1,0)→(0,0,0,0,0,0,0,1,1)→(1,0,0,0,0,0,0,0,0) (back to the first state)

The three states requested can be seen below:

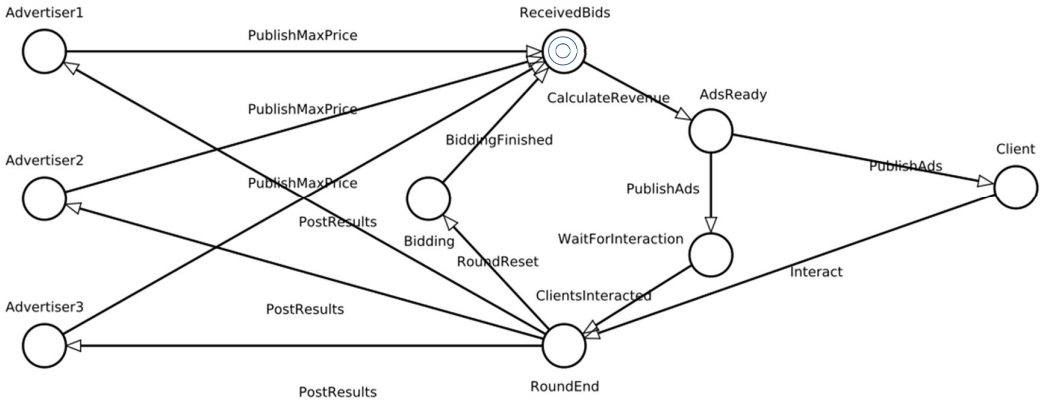
1) The initial state before starting the execution:



2) A mi

The finished round state is the same as the round start because the RoundEnd state is both the start and end state of the round

3) the final state when both bidding rounds have finished:



Phase B

For the phase B we calculated the variables requested in each of the scenarios described:

Scenario A:

Clients

a. The click probability for each position *i* on the ranking list can be calculated using the following formula: $p(i) = \frac{n-i+1}{\sum_1^n x}$ where *i* the position on the list and *n* the amount of list entries.

b. The purchase probability is 50% for all entries in the ranking list because the product is of the same quality.

Advertisers

a. All advertisers are advertising their products with a price 50€.

b. The Advertisers would have the following two strategies:

1. Increase the bid if the clicks are less than the last round
2. Reduce the bid if the clicks are equal or less than the last round

Scenario B

Clients

a. The click probability for each position i on the ranking list is the same as the Scenario A, calculated using the formula:

$$p(i) = \frac{n-i+1}{\sum_1^n x}$$

b. The purchase probability can be calculated using the following formula:

$$p(i) = \frac{180 - price}{2}$$

(The purchase probability will be in range 40-85% for a price in the range of 10-100€)

Advertisers

a. All advertisers are advertising their products with a price in the range 10-100€

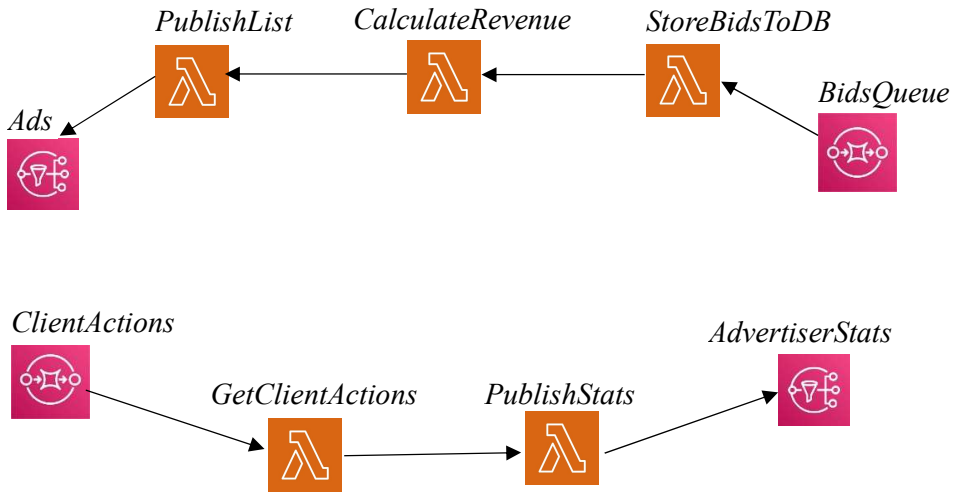
c. The Advertisers would have the following strategies

1. Increase the bid if the clicks are less than the last round
2. Reduce the bid if the clicks are equal or less than the last round
3. Increase price if cost>purchases*price
4. Reduce price if cost<purchases*price

Phase C

C1.

For the ad service provider I created 5 lambda functions, 2 SQSes and 2 SNSes. A simple diagram of these can be seen below with the relations between them as to who calls who.



The code for these lambda finctions can be found in the “PhaseC/lambda functions” directory

Also a DynamoDB table was created named AdServiceDB

C2.

For the advertisers I created 3 EC2 instances containing a python file named advertiser.py. This file creates a Flask server which on startup subscribes to AdvertiserStats SNS. Also on startup initializes a random AdID (0-999999999), Bid (1-100) and Price (10-100). After confirming the subscription the server sends a starting bid to the BidsQueue SQS. Uppon Receiving a Notification containing statistics the advertiser calculates the new bid and price values (price only on the second scenario) and sends the new bid to the BidsQueue SQS while printing its round statistics on the terminal. The advertiser.py file can be found in the PhaseC directory.

C3.

For the client I followed a similar approach as with the advertiser. 10 EC2 instances containing the client.py file which launches a Flask server and subscribes to Ads SNS. The difference with the advertiser is that upon receiving requests, the client picks an ad from the adList it received from the notification based on the probabilities discussed in the PhaseB and also calculates the purchase probability for the clicked advertisement. Finally, the client sends a message containing the information about its choice to the ClientActions SQS and print the round stats in the terminal. The client.py file can also be found in the PhaseC directory.

C4.

For the coordination there are some disadvantages in my approach that I had accept because of the lack of time. The rounds are counted only in the advertisers code, which makes the advertisers kind of coordinators as to when to stop the game.

The other important place where coordination is needed is when SQSes send queued messages.

On the “upper” side of the diagram the ClaculateRevenue lambda function is responsible to wait for all the advertiser bids to be submitted and stored in the database. The way this is achieved is by counting how many table entries exist. This can be acceptable because the table empties every round reset (at PublishStats lambda function). Only if there are sufficient entries in the table the function calls the next one (PublishAds)

On the “lower” side of the diagram the PublishStats lmbda function is responsible for waiting for all the clients to interact. For this to happen there is an entry in the service’s table with AdID=-1 which has a column named Clients_Interracted, where the count of clients is kept.