Example of Use of the Auction Framework for Gridsim

Marcos Dias de Assunção
Grid Computing and Distributed Systems Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
http://www.gridbus.org

This document provides an illustrative scenario for an auction in Grid computing and discusses the design of the auction framework. The example considers a descending Dutch auction that follows the standards provided by FIPA [1,2]. FIPA is a non-profit organization that defines standards for multi-agent systems and for communication among agents in multi-agent systems.

The main participants in an ordinary auction are the seller, the auctioneer and the buyers or bidders. Figure 1 presents an example of reverse auction for Grid computing in which users are buyers, brokers are auctioneers and resource providers are sellers. In reverse auctions, the buyer starts the auction and the sellers bid to sell a service to the buyer. In such a case, a Dutch auction becomes ascending. Hereafter, we use the terms users to buyers, auctioneer to refer to the broker and bidders to resource providers.

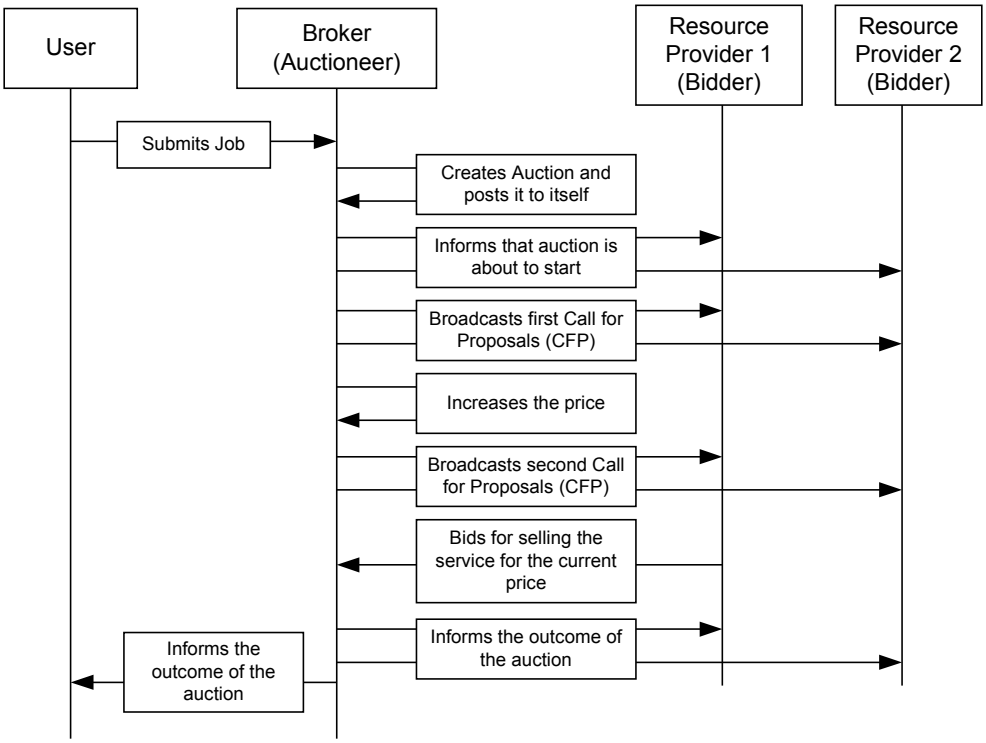

Figure 1. General view of our auction model.

Initially, the user submits jobs to her broker. In the Grid, a broker is responsible for submitting and monitoring jobs on the user's behalf. The broker creates an auction and sets additional parameters of the auction such as job length, the quantity of auction rounds, the reserve price and the policy to be used (e.g. English or Dutch auction policy).

As the broker also plays the role of auctioneer, it posts the auction to itself; otherwise, the auction would be post to an external auctioneer. The auctioneer informs the bidders that a Dutch auction is about to start. Then, the auctioneer creates a call for proposals (CFP), sets its initial price, and broadcasts the CFP to all the bidders. Resource providers formulate bids for selling a service to the user to execute her job. The first time that bidders evaluate the CFP, they decide not to bid because the price offered is below what they are willing to charge for the service. This makes the auctioneer to increase the price and send a new CFP with this increase in the price. Meanwhile, the auctioneer keeps updating the information about the auction. In the second round, a bidder decides to bid. The auctioneer clears the auction according to the policy specified beforehand. Once the auction clears, it informs the outcome to the user and the bidders.

Based on this general model of auctions, we have designed and implemented a generalized auction framework that allows users to develop and evaluate auction protocols for resource management in Grids by using GridSim Grid simulator. Some of the features offered by the current release of GridSim include advance reservation, networking with differentiated services and resources with different allocation policies. The main classes that compose the auction framework are:

- **Auctioneer:** This class extends GridSim entity and implements the basic behavior of an auctioneer. An auctioneer may involve in multiple auctions. The auctioneer sends call for proposals, receives bids, maintains a list of the auctions, and removes them when they are cleared.
- **Auction:** An auction contains basic attributes that are common to every auction.
- **OneSidedAuction:** This class extends Auction and defines methods for auctions that accepts only bids, unlike double auctions. Users may implement different auctions by extending this class and implementing the following methods: onStart(), onClose(), onStop(), onReceiveBid() and onReceiveRejectCallForBid(). These methods have to be implemented to define the behavior of the auction for when it starts a round, when it closes a round, when the auction finishes, when it receives a bid and when the auction receives a rejection respectively.
- **DoubleAuction:** It defines the basic behavior for a double auction. A double auction accepts asks and bids, and tries to match them. Extensions of this class need to implement the methods: onStart(), onStop(), onReceiveAsk() and onReceiveBid().
- **Message:** The class Message provides the basic functionality for messages exchanged by auctioneers and bidders. The framework provides specializations of this class for call for proposals, bids, reject proposal messages, and so on.
- **AuctionObserver:** To participate in auctions, a bidder uses an observer. The bidder could extend the GridSim class and forward messages regarding auctions to the observer, and then the AuctionObserver treats the message and returns the corresponding message to the auctioneer. An observer has a responder, which is responsible for implementing the bidder’s side of the auction policy.
- **Responder:** A class that implements this interface is responsible for defining the bidder's policy. That is, when a bidder receives a call for proposal message, for example, the method onReceiveCfp() of the responder class will be called. The user has to implement the behavior of the responder upon the receiving of different messages.

As example, we present the code for a reverse Dutch auction discussed beforehand:

```
package gridsim.auction;

/**
 * This class represents a Reverse Dutch Auction.
 * In a reverse auction, buyers start the auction and the
 * lowest bid is considered the best. Therefore, the
 * Dutch auction becomes ascending and starts with
 * the min price going until the max price.
 *
 * @author       Marcos Dias de Assuncao
 * @since        GridSim Toolkit 4.0
 * @see gridsim.auction.Auction
 * @see gridsim.auction.OneSidedAuction
 * @see gridsim.auction.AuctionTags
 */
public class ReverseDutchAuction extends OneSidedAuction {
    private MessageBid bestBid;

    /**
     * Constructor
     * @param auctionName a name for the auction
     * @param auctioneerID the GridSim id of the auctioneer
     * @param durationOfRounds simulation time of the duration of each round
     * @param totalRound the number of rounds
     * @throws Exception
     * @see gridsim.GridSim
     */
    public ReverseDutchAuction(String auctionName, int auctioneerID,
                    double durationOfRounds, int totalRound)throws Exception {
            super(auctionName, auctioneerID, AuctionTags.REVERSE_DUTCH_AUCTION,
            durationOfRounds, totalRound);
    }

    /**
     * This method is called when a round is started
     * @see gridsim.auction.OneSidedAuction#onStart(int)
     */
    public void onStart(int round) {
            if (round == 1) {
                    super.setCurrentPrice(super.getMinPrice());
            }

            MessageCallForBids msg = new MessageCallForBids(super
                            .getAuctionID(), super.getAuctionProtocol(),
                            super.getCurrentPrice(), super.currentRound());

            msg.setAttributes(super.getAttributes());
            super.broadcastMessage(msg);
    }

    /**
     * This method is called when the auction finishes
     * @see gridsim.auction.OneSidedAuction#onStop()
     */
    public void onStop() {
            int winner = super.getWinner();
            MessageInformOutcome iout = new MessageInformOutcome(super
                            .getAuctionID(), super.getAuctionProtocol(), winner, super
                            .getFinalPrice());

            iout.setAttributes(super.getAttributes());
            super.broadcastMessage(iout);
    }
```

```
        /**
         * This method is invoked when a round finishes
         * @see gridsim.auction.OneSidedAuction#onClose(int)
         */
        public void onClose(int round) {
                if (round >= super.getNumberOfRounds()) {
                        if (bestBid == null)
                                super.setFinalPrice(super.getCurrentPrice());
                } else {
                        double increase = super.getMaxPrice()
                                        / super.getNumberOfRounds();
                        super.setCurrentPrice((float) super.getCurrentPrice()
                                        + increase);
                }
        }

        /**
         * This method is called when a bid is received.
         * @see gridsim.auction.OneSidedAuction#onReceiveBid(gridsim.auction.MessageBid)
         */
        public void onReceiveBid(MessageBid bid) {
                double price = bid.getPrice();
                super.setFinalPrice(price);
                if (price <= super.getReservePrice()) {
                        super.setWinner(bid.getBidder());
                }
                bestBid = bid;
                closeAuction();
        }

        /**
         * Called when a reject bid is received.
         * @see gridsim.auction.OneSidedAuction#onReceiveRejectCallForBid(
                                gridsim.auction.MessageRejectCallForBid)
         */
        public void onReceiveRejectCallForBid(MessageRejectCallForBid mrej) {
                // do nothing for now...
        }
    }
```

In our example, when the broker receives a job from a user, it starts a Reverse Dutch auction through the following code. See the comments along the code for a better explanation.

```
// creates a Dutch Auction with duration of 60.0 each round
// and maximum of 10 rounds
ReverseDutchAuction da = new ReverseDutchAuction(
        super.get_name() + "_DA", super.get_id(),60.0, 10);

// To make sure that there will be offers to all jobs,
// we set the reserve price to something up to 100
// and offers at 10 maximum
da.setReservePrice((double)(10 + (90 * Math.random())));
da.setMaxPrice(100);
da.setMinPrice(0);

// the auction has a referente to the job/gridlet
da.setAttribute("job",(Gridlet)jobs.get(2));

// set the list of current bidders for this auction
da.setBidders((LinkedList)resourceIDList.clone());

// sends events to auctioneer to post and start the auction
super.send(super.get_id(), GridSimTags.SCHEDULE_NOW, AuctionTags.AUCTION_POST, da);
super.send(super.get_id(), GridSimTags.SCHEDULE_NOW + 120.0,
                AuctionTags.AUCTION_START, new Integer(da.getAuctionID()));
```

The Auctioneer class is extended by the Broker class in our example. Thus, the Broker class needs to implement the method onAuctionClose(), which is invoked when an auction is closed. The code below presents an example of implementation for this method:

```
/*
 * (non-Javadoc)
 * @see gridsim.auction.Auctioneer#onAuctionClose(gridsim.auction.Auction)
   */
   public void onAuctionClose(Auction auction){
       OneSidedAuction osauc = (OneSidedAuction)auction;
       int winner = osauc.getWinner();

       System.out.println("Results of the auction" +
                       "\nWinner ID: " + winner +
                       "\nPrice paid for executing the job: " + osauc.getFinalPrice());

       if(winner != -1){
              gridletSubmit((Gridlet)auction.getAttribute("job"),winner);
       }
   }
```

This code concludes our example about how to use the auction framework. Examples of auctions for traditional auctions such as English, First-Price Sealed Bid and Dutch are provided with the framework.

References

1. Fipa dutch auction interaction protocol specification. FIPA - Foundation for Intelligent Physical Agents (http://www.fipa.org/), August (2001).
2. Fipa english auction interaction protocol specification. FIPA - Foundation for Intelligent Physical Agents (http://www.fipa.org/), August (2001).