PPAPD: Peer-to-Peer Ascending Price Distributed Auction

Tao Lin, Gary Ma

May 8, 2023

Abstract

We implement a peer-to-peer online distributed auction system where participants can be auctioneers and buyers simultaneously, run auctions on their own computers, and involve a service platform for redundancy and fault tolerance but not for running the auction. We use gRPC to implement communication between the auctioneer and buyers, the platform and the auctioneer, the platform and the buyers. We implement RAFT to distribute a state machine of a platform across a cluster of computing system, so as to provide redundancy and fault tolerance to the system. Compared with centralized online auctions conducted by professional auction houses or auction websites, our P2P auction system enables smaller sellers to have more power over how to auction.

1 Introduction

Auctions have ever been an important way by which market participants trade merchandises with each other. The Roman Empire used "atrium auctionarium", a form of ascending price auction, to liquidate property and estate goods, before the empire itself got auctioned off on the auction block in the 193 A.D. after being sacked [5]. The modern days auctions are documented in the late sixteenth century by the Oxford English Dictionary, partly explaining why "English Auction" has been one of the most favorite forms of auctions. After the first documentation in the dictionary, the London Gazette often reported the auctioning of artworks at coffee houses or taverns throughout London in the seventeenth century [2]. And not long after in the eighteenth century, big auction houses including Sotheby's and Christie's were founded.

With the popularization of the internet in the 1980s, auctions have begun to take on a new form: they are now conducted through internet and more sellers and buyers can participate. eBay enables small sellers and vendors to get accessed by auction participants. Once the largest online personal trading community back in the 2000s, eBay still takes up 4.7 percent of market share among online retailers in the U.S as of today [6]. Big auction houses also started to move online. Christie's has developed a sector that only sells through online and this sector have seen rapid growth during the covid years [8]. Besides auctioning off tangible items and artifacts, online advertisement auctions sell advertisement slots on users' browsers, supporting internet powerhouses like Google and Microsoft.

But centralized auctions through the internet have drawbacks. eBay is constantly fending off shell bidding, or being accused of shell bidding themselves [4]. More severely, a centralized platform that both runs the auction and records the auction data can easily manipulate the auction: the U.S. department of justice sued Google for "manipulating auction mechanisms across several of its product to insulate Google from competition, deprive rivals of scale, and halt the rise of rival technology" [3].

In light of the drawbacks of centralized online auction systems, we design and implement a peer-to-peer online distributed auction system. In our system, participants can be auctioneers and

buyers simultaneously and possess controls over their own auctions. A service platform is involved to provide redundancy and fault tolerance but not to run the auction. Because the platform is facilitating the auction instead of fully controlling it, the platform's ability to manipulate the auction is limited. This alleviates the concerns that big companies like Google using auctions to favor their own products.

2 Design

We design an online peer-to-peer (P2P) auction system that implements the modified English auction [1]. There are three parties in our system: a platform, sellers (auctioneers), buyers (bidders). A user logs in the system via the platform and can choose to be either a seller or a buyer. Sellers create auctions and buyers join auctions via the platform. Once an auction is started, all the communication happen between sellers and buyers in a P2P manner, without the platform involved. Nevertheless, the platform can be used to provide redundancy so that users can resume an auction if they crash.

In Section 2.1, we describe the auction format that we implement. Then in Section 2.2, we illustrate the auction system with diagrams in the time order of an auction: before an auction, when an auction starts, during an auction, and when an auction finishes. Our system is built entirely upon remote procedure calls (RPCs). We list all the RPCs in our system in Section 2.3.

2.1 The Auction We Implemented: Modified English Auction

The exact auction format we implement is a **modified English auction**. It is a type of "ascending price auctions" [1] where the price of the item for sale is increased over time by the seller and the bidders bid by "withdrawing":

- Before the auction starts, the seller announces a low base price (e.g., 0 dollar) for the item. Every participating buyer is considered "active".
- Every round, the seller increases the price from x to $x + \varepsilon$, where ε is some minimum price increment (e.g., 1 cent). Each active buyer can choose to "stay in the auction" (continue to be "active") or to "withdraw from the auction" (become "withdrawn"). A buyer not responding is regarded as automatically withdrawing from the auction. A withdrawn buyer cannot re-activate itself in future rounds.
- The auction ends when only one buyer remains active in the auction. The buyer wins the item and pays the current price.[1]

This auction is different from the auction commonly seen in practice, for example in eBay, where in each round one bidder outbids the current highest bid. We choose to implement the modifed English auction instead of the latter one because we do not want to repeat what eBay has done. Also, the modified English auction requires more seller-buyer synchronization than the eBay auction, which makes the implementation more challenging and interesting from a distributed system perspective.

Modified English auction is similar to another auction, called **vanilla English auction**, that is used by major traditional auction houses like Sotheby's. The difference is: in a vanilla English auction, a bidder withdrawing in one round is allowed to rejoin the auction in later rounds. Vanilla

[1] If two or more buyers withdraw in the same last round, the winner might be selected randomly in practice. But in our digital system, we use a lock to prevent buyers from withdrawing at the same time.

English auctions are popular in practice, but from a distributed system standpoint are less interesting than the modified English auctions, because allowing bidders to rejoin the auction makes different rounds of the auction *independent*: bidders and the auctioneer interact in the same way each round except for the price. In contrast, in a modified English auction the auctioneer has to remember which bidders have withdrawn previously and prevent them from rejoining. This inter-dependency between rounds makes the implementation more challenging and interesting.

2.2 Overview of Our System

Now we illustrate with diagrams the component of our auction system. Each diagram depicts agents and their RPCs. An arrow points from the agent initiating the call to the agent who provide the RPC service.

Before an auction Figure 1 illustrates what happens in our system before an auction starts. Users log in the system via the platform server with their usernames, IP addresses, and ports, choosing to be seller clients or buyer clients. The platform serves as a centralized "DNS server" in this stage to store the users' IP addresses and ports. A seller client submits a request to create an auction to the platform, where the request is processed and saved. Buyers can fetch all the auctions on the platform and decide which ones to join. A buyer can switch between joining or quitting in this stage (before auction starts) as many time as it prefers.

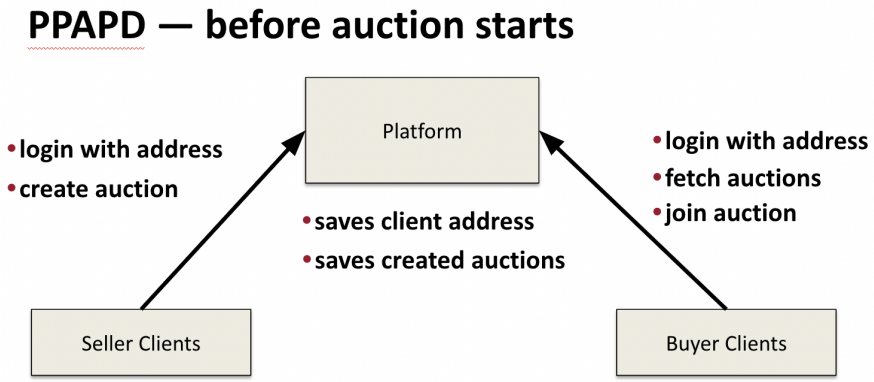

Figure 1: Before an auction

When auction starts This stage is illustrated in figure 2. The seller requests the platform to start the auction. The platform returns the auction information containing which buyers have joined the auction. Buyers and sellers obtain IP addresses (and ports) of each other by fetching from the platform, so that they can communicate with each other directly during the auction.

During and finish auctions This is depicted in figure 3. During an auction, the seller broadcasts the price to every buyer each round and the buyers reply whether to withdraw from the auction or not. The seller also updates the auction information to the platform periodically. The platform saves the auction information, so that the seller and buyers can resume an auction if they crash. When the auction finishes, the seller notifies the platform that the auction has ended.

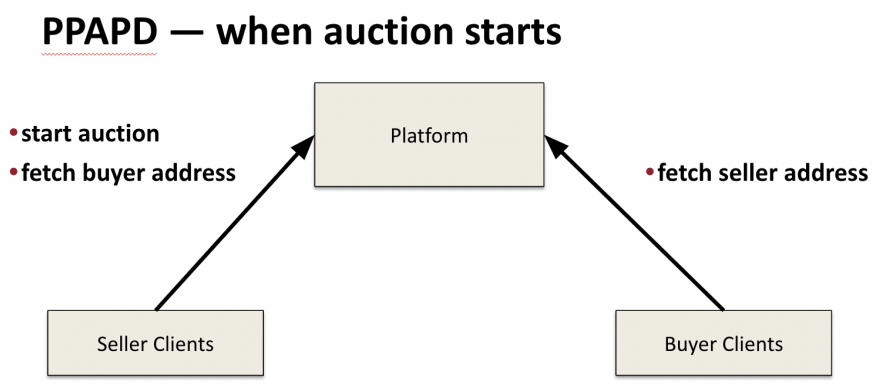

Figure 2: When an auction starts

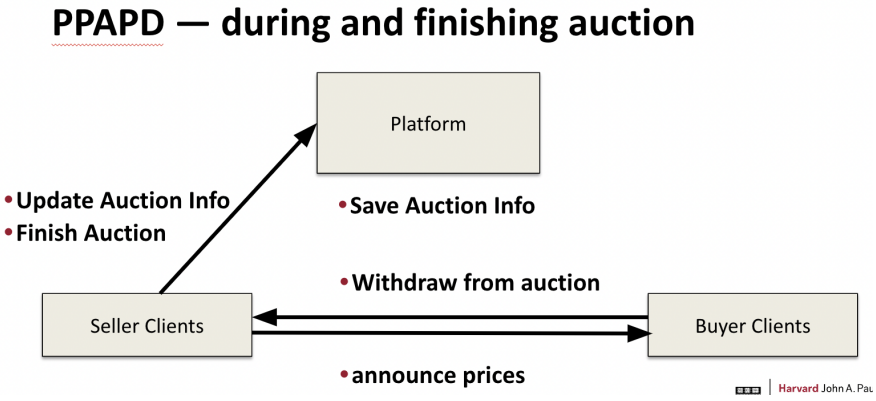

Figure 3: during and finishing auction

RAFT for fault tolerance and redundancy Besides facilitating the auction, the platform records auction information and enables the auction to resume should the seller lose connections. If a seller fails during an on-going auction, the auction is paused. When the seller re-logs in, it fetches from the platform previous auction information that it updated the platform with. The seller can then resume the auction and the buyers will be notified. To guarantee that the platform accurately stores auction information and provides services even with faults, we implement RAFT algorithm on the platform service state machine. Therefore if the leader node fails, new leaders are elected out of candidate nodes and the service continues without disruption.

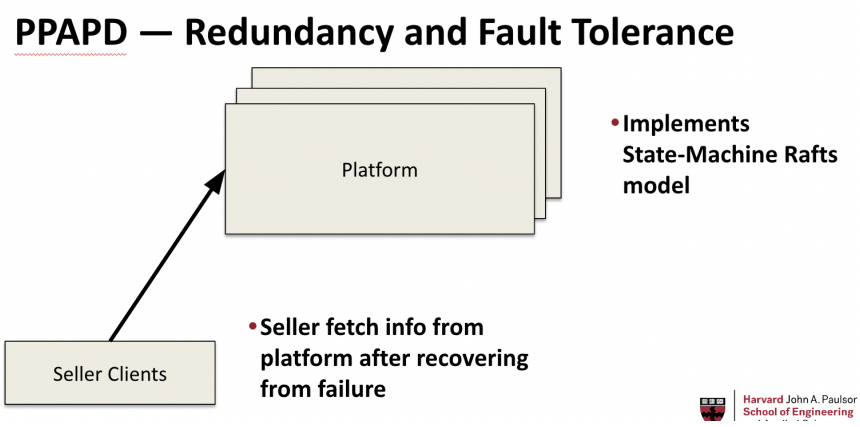

Figure 4: Fault tolerance and redundancy

2.3 List of RPCs in Our System

In this subsection, we describe all the RPCs in our system. Most descriptions are high-level, with some notes explaining important technical details. These RPCs form the basis of our system.

Platform's RPC services The platform provides the following RPC services to a user (as a buyer or a seller):

1. `login(username, RPC_address)` $\rightarrow$ `success_message`:

   A user logs in the system. The user provides its `username` and the `RPC_address` (IP address + port) at which it provides RPC services to other users. The platform records the `RPC_address` and returns success message to the user.

2. `get_user_address(username)` $\rightarrow$ `RPC_address`:

   A user asks for the RPC address of another user, identified by `username`. The platform returns the RPC address of `username`.

The platform provides the following RPC services to a user as a seller:

1. `create_auction(seller_username, auction_name, item_description, base_price, price_increment_period, increment)` $\rightarrow$ `auction_info`:

   The seller (identified by `seller_username`) requests the platform to create a new auction, providing all the relevant information in the RPC request. The platform returns the infor-

mation of the newly created auction, including a unique id of the auction generated by the platform.[2]

2. `start_auction(seller_username, auction_id)` → `(success_message, auction_info)`:

   The seller (identified by `seller_username`) requests to start the auction idendified by `auction_id`. The platform returns whether this operation is successful, and if yes, the information of the auction, including the set of buyers who have joined this auction.[3]

3. `update_auction(seller_username, auction_info)` → `success_message`:

   The seller (identified by `seller_username`) updates the information of a started auction with the platform. The information includes the current price, the current round id, and the set of buyers who have withdrawn from the auction. The seller cannot update information for a non-started or a finished auction. The platform returns whether the operation is successful.

   Note: `update_auction` is not a necessary component of our system. If it is used, then a seller who started an auction and then crashed can resume the auction from the last updated state. If the seller never updated the auction information with the platform, then the auction is resumed from the beginning (i.e., starting from `base_price`).

4. `finish_auction(seller_username, auction_info)` → `success_message`:

   The seller (identified by `seller_username`) notifies the platform that an auction is finished, providing all the information of the auction, including the username of the winner and the transaction price. The platform stores the auction information in its database and returns success message.

   Note 1: The reason that the seller needs to provide all the information of the finished auction in addition to the id of the auction is because the auction information on the platform may be out-of-date.

   Note 2: If the seller calls `finish_auction` multiple times, then starting from the second time the platform will return "success" but not change the auction information in its database. This is because: once an auction is finished, it will be the platform who manages the auction and the seller cannot change the auction information anymore.

5. `fetch_auctions(seller_username)` → a list of `auction_info`:

   The seller asks for the information of all the auctions it owns on the platform.

The platform provides the following RPC services to a user as a buyer:

1. `join_auction(username, auction_id)` → `success_message`:

   A buyer (identified by `username`) requests to join an auction identified by `auction_id`. The auction to join should be not started and not finished. The platform returns "fail" if the auction has started or finished, "success" otherwise.

2. `quit_auction(username, auction_id)` → `success_message`:

[2]In fact, in our implementation, it is not necessary for the platform to return the information of the newly created the auction if the seller periodically fetches the information of all the auctions it owns from the platform.

[3]The reason that the platform needs to provide this information to the seller in addition to the success message is that, before started, the auction is managed by the platform and the the seller may not know which buyers have joined the auction. See Section 4 for a detailed discussion of data ownership transfer in our system.

A buyer (identified by `username`) requests to quit from a (not started and not finished) auction identified by `auction_id`. The platform returns "fail" if the auction has started or finished, "success" otherwise.

3. `fetch_auctions(username)` → a list of `auction_info`:

   A buyer (identified by `username`) asks for the information of all the auctions on the platform. For each auction, the platform provides all information of the auction (including the current price, round id, set of participating buyers) to the buyer if the buyer has joined the auction. If the buyer has not joined the auction, only basic information (like base price, item description, not current price) is provided.

Buyer's RPC services A buyer provides the following RPC services to a seller:

1. `announce_price(auction_id, round_id, price, buyer_status)` → `acknowledgment`:

   A seller calls this service to tell a buyer the latest information of an auction (identified by `auction_id`), including the current price (`price`), the current round (`round_id`), and the withdrawn/active status of all buyers in the auction. The buyer acknowledges that it receives this RPC request.

2. `finish_auction(auction_id, winner_username, price, buyer_status)` → `acknowledgment`:

   A seller calls this service to tell a buyer that an auction (identified by `auction_id`) is finished, providing the information of the winner (`winner_username`), the transaction price (`price`), and the status of buyers in the auction. The buyer acknowledges that it receives this RPC request.[4]

Seller's RPC services A seller provides the following RPC service to a buyer:

1. `withdraw(username, auction_id)` → `success_message`:

   A buyer (identified by `username`) requests to withdraw from an auction (identified by `auction_id`). The seller tells the buyer whether the `withdraw` operation is successful or not. (For example, if the buyer is the only active buyer in the acution, then the buyer should be the winner and cannot withdraw. And a buyer cannot withdraw from an already finished auction.)

   Note 1: If a buyer requests to withdraw multiple times and the first time is successful, then all the following times are also successful. This makes the operation idempotent and simplifies the design.

   Note 2: If the number of active buyers is reduced to 1 after a `withdraw` operation, the seller should immediately finish the auction (i.e., set the auction status to "finished" and notify the platform and the buyers in the auction by their `finish_auction` RPCs).

3 Demonstration

We select three cases to demonstrate the fault tolerance and redundancy of our system. A success trial for baseline, a seller crash trail and a server crash trail for testing fault tolerance and redundancy. See github readme.md for details.

[4]This RPC service is actually not necessary in our system: once an auction is finished, the seller will notify the platform, and then the buyers can know that the auction is finished by periodically fetching auction information from the platform. Nevertheless, this RPC service allows the seller to notify the buyers that an auction is finished even when the platform server is down.

4 Discussion

4.1 Design Choices

This subsection discusses some questions we had and choices we made in our design.

Why P2P? We decided that, once an auction is started, the seller and the buyers should interact directly without using the platform as an intermediary, i.e., in a P2P manner. Besides the reasons mentioned in Section 1, another reason for implementing a P2P system is *efficiency*. Suppose there are a million sellers in the system each updating prices 100 times per second to 10 buyers, then a server acting as an intermediary will have a huge workload ($10^6 \times 100 \times 10$ operations per second), which is undesirable. This workload is even huger if the server operations are replicated. Instead, if we let the sellers announce price to the buyers whenever the price changes and update the auction information with the platform only once every second, then we can reduce the server's workload to 10^6 operations per second.

Why do we make the joining and quitting operations not P2P (buyers ask the platform to do these operations, not the seller)? Because in this way the buyers can join and quit an auction even when the seller is offline.

How to tell apart buyers' actual withdrawals from the auction and network faults? This is not a problem in offline auctions where buyers never get disconnected. In the online setting, a buyer who accidentally gets disconnected with the seller for a short time should not be considered as withdrawing from the auction and should continue to be "active" after reconnecting. But if we allow a buyer who is disconnected for multiple rounds to re-activate, then this auction essentially becomes a vanilla English auction where bidders can rejoin the auction at any time. So, we decided that a buyer disconnected for more than one round is considered as withdrawing, and a buyer who disconnects and reconnects within a round as not withdrawing. To implement this, the seller requires an acknowledgment from every buyer when announcing price change to them at every round (using the `announce_price` RPC). If a buyer does not acknowledge, this buyer will be automatically withdrawn by the seller.

4.2 Interesting Features of Our System

Our design has some interesting features:

The auction platform serves as a "DNS server". Due to our P2P design mentioned above, the buyers and the sellers need a way to know each other's IP address (and port) in order to communicate. To do this, we let our auction platform be a "DNS server": when a user logs in, the user client automatically sends the username together with the IP address and port to the platform, who records this in its database. When a user A wants to communicate with another user B, it provides the username of B to the platform to ask for B's IP address and port. When a user re-logins with a different IP address and port, the platform will update that information in its database, so our system allows users to change IP addresses as well.

The states of an auction serve as logical clocks. The "created", "started", "finished" status of an auction and the round id of an started auction serve as "logical clocks" [7] in our distributed system. For example, if the platform receives an RPC from a seller requesting to start an auction

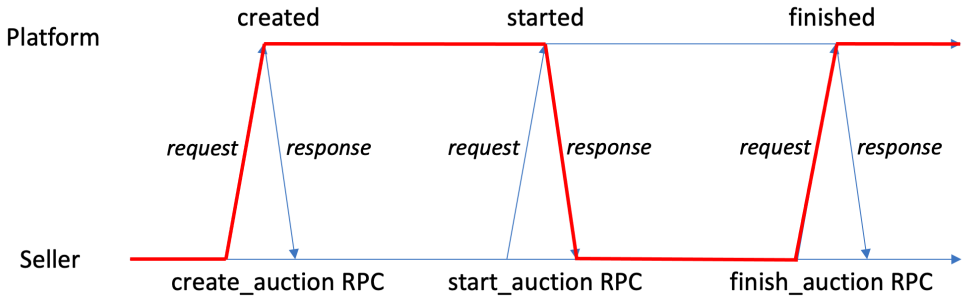

Figure 5: Transfer of auction data ownership

that is already finished in the platform's record, then the platform should consider this request as outdated and reject it. If a buyer in an started auction first receives an `announce_price` RPC request from the seller with (`round_id = 10, price = 100`) and then receives one with a smaller round id, e.g., (`round_id = 9, price = 90`), then the buyer should ignore the second request. (This scenario can happen if the price is updated frequently and the two RPC requests reach the buyer out of order.)

The ownership of auction data is transferred between platform and seller. Figure 5 illustrates the transfer of ownership of auction data in our system. An auction is initially created by the seller. Once created and before started, it is the platform who manages the auction. Buyers join or quit the auction by asking the platform, not the seller. Once the seller requests the platform to start an auction, the platform gives all the information of the auction to the seller. The seller then manages the auction until it is finished. Buyers make `withdraw` requests to the seller instead of the platform. When the seller requests the platform to finish an auction, the seller sends the auction data back to the platform and the ownership is transferred to the platform *permanently*. No further update or finish auction requests from the seller are allowed once the auction is finished.

As the auction data are maintained by both platform and seller, care must be taken when we synchronize the data between the two parties. In particular, when fetching auction data from the platform (using the `fetch_auctions` RPC), the seller should replace its data with the platform's data for not started auctions and finished auctions, and ignore the platform's data for started and not finished auctions.[5] On the contrary, when the seller updates auction data to the platform (using the `update_auction` RPC), the platform should only update started and not finished auctions.

References

[1] *Ascending Auctions*. Lecture note of CS1951k, Brown University. 2020. URL: `https://cs.brown.edu/courses/cs1951k/lectures/2020/ascending_auctions.pdf`.

[2] Ralph Cassady. *Auctions and auctioneering*. Univ of California Press, 1967.

[3] US Justice Department. *Justice Department Sues Google*. 2023. URL: `https://www.justice.gov/opa/pr/justice-department-sues-google-monopolizing-digital-advertising-technologies` (visited on 01/24/2023).

[5] "Not started", "finished" according to the platform's data.

[4] Ebay. *Shill bidding policy— eBay*. 2023. URL: https://www.ebay.com/help/policies/selling-policies/selling-practices-policy/shill-bidding-policy?id=4353 (visited on 2023).

[5] Edward Gibbon. *The Decline and Fall of the Roman Empire*. Delmarva Publications, Inc., 2015.

[6] JUOZAS KAZIUKĖNAS. *eBay's Market Share Slips to Below 5 percent*. 2022. URL: https://www.marketplacepulse.com/articles/ebays-market-share-slips-to-below-5 (visited on 02/24/2012).

[7] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system". en. In: *Communications of the ACM* 21.7 (July 1978), pp. 558–565. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/359545.359563. URL: https://dl.acm.org/doi/10.1145/359545.359563 (visited on 05/07/2023).

[8] Statista. *Total online auction sales of Christie's worldwide from 2017 to 2023*. 2023. URL: https://www.statista.com/statistics/999436/christie-s-online-only-sales/ (visited on 04/18/2023).