Make restaurants pay your server bills

Grubenmann, Tobias ; Dell' Aglio, Daniele ; Bernstein, Abraham ; Moor, Dmitrii ; Seuken, Sven

Make restaurants pay your server bills

Tobias Grubenmann, Daniele Dell'Aglio, Abraham Bernstein,
Dmitry Moor, and Sven Seuken

Department of Informatics, University of Zurich, Switzerland,
{grubenmann, dellaglio, bernstein, dmitry.moor, seuken}@ifi.uzh.ch

Abstract. So far, the Web of Data (WoD) has only marginally considered the problem of financing itself. In most of the cases, research funds (either public or private) and donations are supporting the creation and maintenance of WoD services. Applying typical Web financing mechanisms to the WoD is not straightforward, since the machine-processable nature of the WoD introduces new challenges. For example, users may access the data through applications that filter out advertisement. We believe that delayed-answer auctions are a possible way to overcome such a problem: when a user submits a query, the SPARQL endpoint serves the answers in sequence of batches. Behind the scenes, sponsors bid to increase the chances that data they are interested in promoting appears at the beginning of the sequence. In this demo, we show how this delayed-answer auction can be realized. We built a scenario where users can query for restaurants in New York City and restaurant owners act as sponsors.

Keywords: Web of Data · Slot Auctions · Delay · Vickrey-Clarke-Groves mechanism

1 Introduction

The Web of Data (WoD) consists of interlinked data owned and provided by different entities. The result is one giant, distributed RDF graph which stores knowledge related to all different kinds of topics. On a technical level, the WoD has successfully extended the World Wide Web (WWW) idea, enabling the integration and querying of distributed and heterogeneous graph data. On an economical level, however, key concepts and techniques to finance these new services are lacking behind [2]. One reason for this is that the classical financial motors of the WWW—advertisement and donation campaigns—cannot be straightforwardly applied to the WoD. The problem is that the machine-processable nature of the WoD renders these approaches useless as they rely on users seeing them while navigating the Web sites. To remedy this situation, we proposed in [1] the *delayed-answer auction*, a mechanism where third-party sponsors invest into the publishers of RDF data. In this paper, we demonstrate an implementation of the theoretical concepts from [1]. As a setting, we consider users searching for restaurants in New York City through a WoD-based service.

2 The Delayed-Answer Auction

The core idea of the delayed-answer auction is to delay parts of a SPARQL answer to influence the *probability* that a certain solution set is considered by the user issuing the query. Given solutions composing a SPARQL query answer, each of them has a different delay. Consequently, the solutions are not delivered in one batch, but are delivered in a sequence of batches at different time. Naturally, the sooner a solution is provided to the user, the higher the probability that the user considers it. We call *sponsors* the entities interested in minimizing the delays of certain solutions. However, a sponsor is not just promoting any arbitrary solution, but solutions which contain *service links*, i.e. URIs to specific services or Web pages. Examples of sponsors are hotel and restaurant owners, product shops or touristic departments of cities. The delayed-answer auction allows sponsors to increase the chance that a user will consider buying their service, similar to advertisement in the WWW. Unlike advertisement, however, data containing service links are legit solutions to the SPARQL query and do not constitute additional and, potentially, distracting information.

The design of the auction mechanism relies on two key questions: how to assign the delays and how to charge the sponsors. To decide which solution should receive which delay, the mechanism ranks the different solutions according to the bids that the included service links receive. A solution containing a service link with a high bid is ranked before a solution containing a service link with a lower bid. This ranked list is used to decide the order on which solutions are returned. It is worth noting that the auctioneer can decide to perturbate this ranked list to allow solutions containing low (or missing) bids to also receive a top position, although with a lower probability. To increase the comprehensibility of our demo, we did not implement such perturbations, however.

In our delayed-answer auction, sponsors only pay if a user actually looks up the corresponding service link. To determine the actual price a sponsor has to pay when the service link is looked up, we use the pricing mechanism of the weighted Vickrey-Clarke-Groves (VCG) auction [3]. The advantage of using this pricing mechanism is that the auction is robust against manipulations of a single sponsor: a sponsor is not able to influence the utility (weighted difference of value and price) by changing the bid [4]. The VCG price π_i of a service link i can be calculated using the following formula:

$$\pi_i = \frac{\sum_{j=b(i)}^{m} v_j^{\text{2nd}} \cdot \Delta p_j \cdot \frac{1-(1-p_{\text{rel}})^{N_j}}{N_j}}{\sum_{j=b(i)}^{m} \Delta p_j \cdot \frac{1-(1-p_{\text{rel}})^{N_j}}{N_j}},$$

where $b(i)$ refers to the batch number of link i—counting starts from the batch with the lowest delay—and m refers to the total number of batches. N_j is the sum of solutions contained in the batches from 1 to j and p_{rel} refers to the probability that a specific solution is relevant to the user—this probability is the same for all solutions and has to be estimated by the auctioneer when setting

up the auction. The parameter Δp_j indicates how likely it is that the user stops waiting for more solutions after receiving batch j and selects one of the received solutions, or abandons the search. Again, the Δp_js have to be estimated by the auctioneer when setting up the auction. Finally, v_j^{2nd} indicates highest reported bid in the next batch after batch $b(i)$[1].

It is worth stressing the importance of the delays to create the necessary *competition* to incentivize sponsors to bid money. Without them, there would be a very limited influence on the order of the solutions and, consequently, on the probability that a user follows up a certain service link. This is because the inherent structure of a SPARQL query answer allows an application to easily reorder any orderings of the solution one might impose.

3 Demonstration

To show the feasibility of the delay-answer auction and how it works, we built a demo[2]. The setting is the one of a search engine for restaurants in New York, built on top of some open data service. We used Blazegraph as a triple store and OpenStreetMap[3] to collect the restaurant data (converted to RDF).

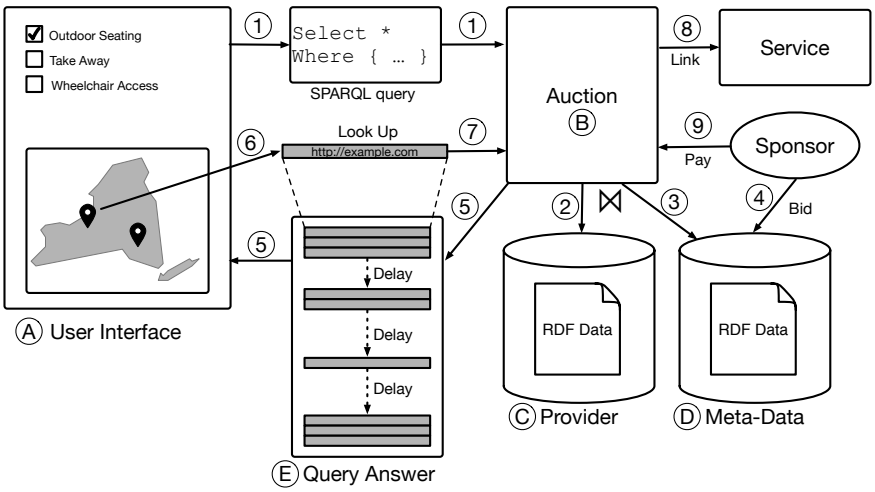

Fig. 1. Overview of the workflow for our delayed-answer auction.

Figure 1 offers an overview of the demo. The Web page shows a search mask where the users can choose among different selection criteria (Ⓐ in the figure), such as outside seating, wheelchair access and the possibility to take away. Once

[1] Interested readers can find in [1] additional details and insights.
[2] `https://files.ifi.uzh.ch/ddis/delay-auction-demo/`
[3] `https://www.openstreetmap.org`

the user submits the search request, a SPARQL query is generated and issued to the auction platform (①). The auction platform (Ⓑ) generates a query answer by executing the query (②) on the data of all participating data publishers (Ⓒ). In addition to the actual data needed to answer the SPARQL query, the auction platform queries an additional triple store (Ⓓ) holding meta-data necessary for the auction (③). These meta-data include the service-links—the service links need to link to the auction which redirects the user to the actual service—and the bids placed on them (④). In this demo we assigned a uniformly distributed random bid between 0 and 1 to each service link. The solution sets of the query answer have to be sorted (in descending order) according to these bids. This is done during query execution using the `ORDER BY` clause of SPARQL.

After the different solutions have been ordered according to the bids, the query answer (Ⓔ) starts to be sent to the user (⑤). The solutions are grouped into *batches*: a batch is a set of solutions which have the same delay. The delay periods are decided by the auction designer when setting up the auction. The transmission of the answer is therefore asynchronous, and it ends when no more solutions are left, or the user cancels the search. The demo adopts a polling strategy, but pushing strategies are viable as well.

The user interface displays the received solutions on a map, including all available data and the service link. The user can browse through the received results and eventually click on one of the service links (⑥) or abandon the search. Clicking on a service link redirects the user to the auction platform (⑦), which in turn, redirects the user to the booking page of the restaurant (⑧). This redirection is necessary for the auction platform to register the look up of the service link. Once such a look up is registered, the sponsor is charged the respective price (⑨).

As we have seen, using the delayed-answer auction, the auctioneer can generate revenue from the payments of the sponsors. In turn, part of this revenue is invested into the datasets of participating data publishers, which solves our problem of financing datasets in the WoD.

Acknowledgments: This work was partially supported by the Swiss National Science Foundation under grant #153598.

References

1. Grubenmann, T., Bernstein, A., Moor, D., Seuken, S.: Financing the web of data with delayed-answer auctions. In: WWW 2018: The 2018 Web Conference (2018)
2. Grubenmann, T., Dell'Aglio, D., Bernstein, A., Moor, D., Seuken, S.: Decentralizing the Semantic Web: who will pay to realize it? In: Proceedings of the Workshop on Decentralizing the Semantic Web (DeSemWeb) (2017), `http://ceur-ws.org/Vol-1934/contribution-01.pdf`
3. Nisan, N., Ronen, A.: Computationally feasible vcg mechanisms. Journal of Artificial Intelligence Research 29, 19–47 (2007)
4. Varian, H.R., Harris, C.: The vcg auction in theory and practice. American Economic Review 104(5), 442–45 (2014)