QueryMarket Demonstration: Pricing for Online Data Markets

Paraschos Koutris, Prasang Upadhyaya,
Magdalena Balazinska, Bill Howe, and Dan Suciu
University of Washington, Seattle, USA
{pkoutris,prasang,magda,billhowe,suciu}@cs.washington.edu

ABSTRACT

Increasingly data is being bought and sold online. To facilitate such transactions, online data marketplaces have emerged to provide a service for sellers to price views on their data, and buyers to buy such views. These marketplaces neither support the sale of ad-hoc queries (that are not one of the specified views), nor do they support queries that join datasets. We present QueryMarket, a prototype data marketplace that automatically extrapolates prices to ad-hoc queries, including those with joins, from the manually priced views. We call this capability "query-based pricing" and describe how it is superior to existing pricing methods, and how it provides more flexible pricing for the sellers. We then show how QueryMarket implements query-based pricing and how it generates explanations for the prices it computes.

1. INTRODUCTION

Data is becoming a traded commodity. Commercial interests increasingly purchase data online to facilitate market research, to inform business decisions, or generate mashups [3]. For example, Xignite [13] sells financial data, Gnip [1] sells data from social media, and AggData [2] collects and sells multiple kinds of online data. The rising demand for valuable online datasets has led to the emergence of various online data marketplace services such as Windows Azure Marketplace [3] and Infochimps [7] that provide a common platform for data owners to sell their data. The demand for such services can be gauged from the fact that Windows Azure Marketplace offers over 130 data sources from 48 publishers in 17 categories while Infochimps offers over 12,000 datasets from multiple vendors.

To use services such as the Windows Azure Marketplace or Infochimps, data sellers need to define prices for their datasets. Right now, these platforms offer limited options for pricing data: the sellers have to specify in advance which (parameterized) views they want to sell and at what price. For example, *www.wordfrequency.info* (explained in detail in Section 2) sells a basic word-list for $195, the word-list augmented with frequencies for $265, and the word-list augmented with co-located words for $265. This is a serious limitation: the seller needs to carefully design the views to sell so as to cater to as many buyers as she can, without knowing the exact queries from the buyers. Similarly, the buyer needs to carefully evaluate the various views that are available for sale to determine which views suffice to answer her query and are also the cheapest way to answer that query. Finally, is not possible today to directly buy queries that join datasets from different sellers.

In prior work [8], we studied the theoretical foundations of assigning a price to an arbitrary query by extrapolating the prices that sellers set on views consisting of selection queries on a single attribute of a relation (note that all the tuples that satisfy the selection predicate are returned and not just the column being selected). We called this capability "query-based pricing." In this demonstration, we apply those results and consider the practical constraints involved in building a commercial marketplace. We present a prototype data marketplace powered by query-based pricing called QueryMarket. QueryMarket is a cloud-based service for data publishers to sell data and ad-hoc views over that data. QueryMarket allows sellers to assign prices to views defined by selection predicates of the form $\sigma_{a=\alpha}$ and allows buyers to purchase any query in a non-trivial subset of *full conjunctive queries without self-joins* [8] called the "Generalized Chain Queries" (Section 5), a set that includes all path join queries, star join queries, and their combination.

Our approach to query-based pricing works as follows: Given a query, the algorithm selects the least expensive set of *answer views* that can be used to answer the query. The sum of the price of these views is the price of the query. This pricing function has two properties: it is *arbitrage-free* (there is no less expensive way to answer the query), and it is *maximal* (no unintended discounts are introduced). The key technical contributions of this prior work include (1) an algorithm with polynomial time data complexity for computing the price of any generalized chain query, by reducing the problem to network flow, (2) a complete characterization of the class of Conjunctive Queries without self-joins that can be priced with PTIME data complexity (this class is slightly larger than generalize chain queries), and (3) a proof that pricing all other queries is NP-complete, thus establishing a dichotomy on the complexity of the pricing problem when all views are selection queries. For the queries that are not in PTIME, we reduce the problem to an Integer Linear Program and compute the prices.

Designing a data marketplace service with the ability to optimally price ad-hoc queries presents three new challenges beyond the theoretical problems that we addressed in prior work: (a) First, since users may ask for prices of queries before actually buying them, the price computation needs to be in near real-time. (b) Second, because setting prices in a way that ensures good properties (*i.e.*, no arbitrage) can be difficult for sellers, a marketplace service should help those sellers identify and fix pricing problems. (c) Finally, since the answer set of views used to price a query can be large, there should be a way to explain to the buyers how the price was actually computed.

In the face of these challenges, this demonstration makes the following contributions:

- We demonstrate the benefits of our query-pricing approach through use-cases on real data. In particular, we enable the audience to interactively compare the pricing and expressiveness of our system with other pricing systems where only pre-defined views may be bought, such as buying the whole dataset for a fixed price and buying per transaction (where a transaction is defined as a fixed number of tuples in the answer). We show that our system can answer more complex and informative queries over the dataset, and can also compute the minimum arbitrage-free price for a given query automatically and in near real-time (Section 2).
- We develop a new method to help sellers fix inconsistent prices and we demonstrate this approach interactively using the real use-cases. We show that QueryMarket not only provides increased flexibility in pricing the input data but also guarantees that the input prices themselves do not violate the arbitrage-free property. (Section 3).
- We develop a new approach for illustrating how QueryMarket prices queries and use this approach to better explain the internals of our system to the audience. More precisely, we demonstrate how QueryMarket computes arbitrage-free answers to ad-hoc queries and provides an intuition to buyers as to why the query has the price it computed. Specifically, it lets buyers ask what all views are required to prove the existence or justify the absence of a certain tuple in the query's answer (Section 4) and compares them agains the views purchased by a naive pricing solution.
- We present a simple prototype implementation of the theoretical framework developed in our recent work (Section 5).

2. BENEFITS OF QUERY-BASED PRICING

The first contribution of this demonstration is to show the QueryMarket prototype in action on real data and to illustrate the benefits of our query-based pricing approach. We describe this part of the demonstration in this section.

Demonstrated Scenario. Microsoft Azure Marketplace has a large online database of English words with their translation into a variety of other languages [9], available in chunks of $20 for 2 million characters (but the first 2 million characters are free to translate). The Website *www.wordfrequency.info* [12] holds a database of 60000 English words along with their parts of speech and frequencies in different genres of documents in a large corpus (*e.g.*, medical documents, movies, financial news articles, etc.). This dataset is available for $265. *www.wordfrequency.info* also sells the most frequently co-located word (a word found immediately before or after a given word) for 60000 English words in the document corpus, also for $265.

We consider a potential buyer, Alice, who is preparing for an interview as an English-to-Greek translator for the World Health Organization. She wants to buy the English-to-Greek translation of all the nouns in the 100 most-commonly used English words in the medical literature. With the prices that are set so far, she must buy the whole word frequency dataset and additionally pay for the translation of the words she is interested in for a total of $265. But that is too much money for her so she chooses to buy nothing. The sellers just missed the opportunity to make a sell.

Now, consider a commercial marketplace service where the seller can sell individual tuples, such as the Windows Azure Marketplace[1]. Alice could get her query answered for a lesser price, but she has to know enough about the two datasets to design and execute a cost-effective plan. For example, she could buy the 100 most frequent words, select those that are nouns, and buy their Greek translations. Not only are these plans cumbersome to generate manually, have to be orchestrated outside the marketplace, and possibly require the use of a language apart from SQL (to automate), but they can also become complex as buyers are given more choices to query the dataset. For example, the seller may also sell just the nouns in different genres and depending on the relative value of the prices of top 100 words' list versus the prices of the noun list, it might be cheaper to only buy the noun dataset for the medical genre, select the nouns with a rank in the top 100, and then buy their translations.

Technical Contribution. One can generalize the individual tuple pricing, mentioned in the above example, by allowing the seller to set a price for selection queries on various attributes of the relations for sell. For example, the seller can allow Alice to buy just the word with rank n where n could be any rank from 1 to 60000, or all words in the genre g where g could be any of the available genres, etc. Given these choices, our system, QueryMarket, provides built-in support for finding the cheapest price to answer a large subset of *full conjunctive query without self-joins.* To show our approach's effectiveness, we compare its price to those computed by Infochimps and Microsoft Azure Marketplace.

Demonstration. We now describe the datasets and the queries used in the demo. The three *paid* datasets are:

- English word frequent dataset: We denote this dataset with the relation F with schema *(English-word, type, genre, rank).*
- Translation dataset: We denote it with the relation T with schema *(English-word, Greek-word).*
- Co-location dataset: We denote this dataset with the relation C with schema *(English-word, English-word, rank).*

[1] Windows Azure Marketplace restricts the seller to price all tuples the same, while the seller might want to have variable pricing. For example, *www.wordfrequency.info* might want to sell the top 100 words at a higher price than the next 100 words, and so on.

We consider three queries that Alice might ask.

Q_1 "The 100 English words that appear most frequently in documents related to medicine." This translates to the following DataLog query:

$$Q_1(e,t,g,r) \text{ :- } F(e,t,g,r), g = \text{'med'}, r \leq 100$$

Q_2 "The Greek translations of the nouns that occur in the list of the 100 English words that appear most frequently in documents related to medicine." This translates to the following DataLog query:

$$Q_2(e,h,t,g,r) \text{ :- } T(e,h), Q_1(e,t,g,r), t = \text{'n'}$$

Q_3 "The Greek translations of the *three* most frequent words collocated with the nouns that occur in the list of the 100 English words that appear most frequently in documents related to medicine."

$$Q_3(c,h,f,e,t,g,r) \text{ :- } T(c,h), C(e,c,f), f \geq 3, \\ Q_1(e,t,g,r), t = \text{'n'}$$

In the demonstration, we use parameterized versions of the above queries to enable members of the audience to experiment with different query variants by interactively choosing different genres, cut-off ranks, and word types.

In our demonstration, we compare QueryMarket's prices for the three queries, Q_1, Q_2, and Q_3, with the prices obtained from a sequence of pre-defined query plans that can be used with existing data market places: (a) *Naive:* buy all datasets, (b) *Rank:* buy the rank column completely, and then only ask for the join-able tuples from the other relations, (c) *Type:* buy the noun column completely, and only then ask for the join-able tuples from the other relations, and (d) *Genre:* buy the genre column completely and then only ask for the join-able tuples from the other table.

In the demonstration, the audience will pick any of the above three queries and will select values from a drop-down menu for the different parameters (with default values already filled in). Additionally, to compare QueryMarket with these pre-defined query plans, the audience will also select from a series of three pre-defined prices where either the ranks, the types, or the genres are the least expensive to buy compared to the other two columns. We then present the price computed by QueryMarket and the other query plans for all the different pricing points.

Overall, in this part of the demonstration, we thus show that our system enables a user to automatically obtain the cheapest price by simply asking her query directly. Second, we show that the prices computed by QueryMarket are better than *all* the other pricing plans we considered. Finally, we show that QueryMarket computes the prices quickly enough to be interactive.

3. SETTING PRICES

In the second part of our demonstration, we explain to the audience how QueryMarket helps users verify the correctness and fix their prices.

Technical Contribution. We consider the prices set by the seller to be inconsistent when there is an arbitrage amongst the prices, that is, a priced selection view on a column is priced so high that it is cheaper to buy all the selection views on some other column and thus get the high-priced column. Not only does QueryMarket check for the consistency of the prices, it also provides suggestions about updates that would make the prices consistent.

We use a lemma [8, Lemma 3.1] to check that prices a seller sets are consistent. The lemma uses the notion of *fully-covered* columns in a relation that are columns where the selections on all values in the column's domain have been priced. The prices are consistent if and only if there is at least one fully-covered column in each relation, and no price assigned to a single selection query, from a relation R, exceeds the sum of the priced columns of any fully-covered column, also from relation the R. This requirement can be easily expressed as a set of SQL queries and it returns the set of prices that lead to inconsistency for free.

Demonstration. As part of the demonstration, we enable the audience to act as the seller and assign prices to selection queries on relation attributes using one of the following two methods:

- Choose from a set of pre-defined functions such as uniform prices (*i.e.*, a selection query that asks for all words in genre "medicine" costs the same as a query that asks for all words in genre "technology", or "law", or other) or prices proportional to the value of the selection attribute (*e.g.*, a query that selects all words with rank value 1 is more expensive than a query which selects all words with rank value 2, etc.).
- Manually set the price of a selection query for a given attribute value (*e.g.*, change the price of the selection query for all words with rank 1 to be 10X higher than before).

We then demonstrate the detection of inconsistent pricing. Either the audience member will cause an inconsistency or we will make a pre-defined consistent pricing on the Frequency dataset inconsistent, by updating the price of a rank to a very high value.

In this part of the demonstration, we will thus illustrate some of the challenges that can arise when manually setting data prices and we will show how QueryMarket helps users avoid such problems.

4. EXPLAINING PRICES

In the last part of our demonstration, we explain to the audience how QueryMarket computes the prices of queries posed by the buyers and how it enables buyers to understand the computed prices.

Technical Contribution. Given a query to price, QueryMarket uses our dichotomy theorem [8], to check, *without executing the query*, if the query's price can be computed in PTIME. If the query's price can be efficiently computed, it computes the price and returns the price to the buyer.

QueryMarket also explains why a query should have the computed price. Unlike existing data marketplaces, QueryMarket provides more flexible pricing and, thus, requires an interface to explain query prices to the buyers.[2] Through this module, the buyer can ask details about a query's price, which is the sum of the selection views bought to answer the query. Such details can be one of the following:

[2] Note that existing marketplaces have very simple billing mechanisms since they offer a very limited query interface.

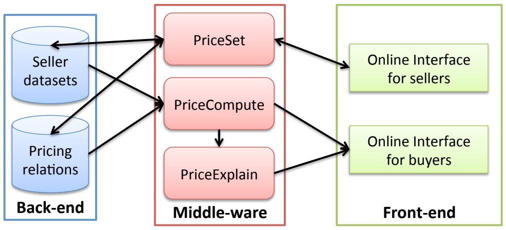

Figure 1: High level architecture of QueryMarket. The direction of the arrows represent direction of data transfer.

- For a tuple in the query's output, the user can request the views that were purchased to generate the tuple.
- For a tuple absent from the query's output, the user can request the views that justify the tuple's absence.
- To give an intuition for why a view was purchased or not purchased we compare against the views purchased by the naive pricing scheme that consists of buying each relation individually at the cheapest price.

Demonstration. Given a query to price, such as Q_2, we check if it is admissible, and if it is, we compute its price. Since the flow graph to price the data can be larger than the size of the database, it is not possible to show it visually for the use-case we have outlined previously. Instead, we provide hard-coded examples of the flow graph of the three queries on a very small dataset.

We provide an online interface for the buyers to understand the price of their queries. For example, to answer Q_2, the buyer can ask QueryMarket which view justifies the tuple with rank 1 (by typing out the tuple). Similarly, the user may ask which purchased view proves that a certain tuple (by typing out the tuple) was not in the answers of Q_2. We also let the users compare the views purchased by the naive pricing strategy to the view purchased by QueryMarket. We let users to focus on different parts of this set of views (sort by relations, sort by attribute value, sort by prices, etc.).

5. PROTOTYPE DETAILS

Figure 1 illustrates the architecture of the QueryMarket cloud service. QueryMarket provides an web interface for sellers to upload datasets and set prices to selection queries over columns of these datasets. It also enables buyers to query the uploaded datasets and pay for the result. The price paid is automatically derived by QueryMarket based on the price points specified by the sellers.

QueryMarket is implemented as a middle-ware layer on top of a traditional relational database management system. The key components of the system include:

- *PriceSet*: Enables sellers to set price-points. Detects when price-points conflict and lead to arbitrage opportunities. Provides recommendations to users for adjusting their price-points.
- *PriceCompute*: Given the price-points specified by the sellers, it computes the price of the queries submitted by the buyer. If a price of a query cannot efficiently be computed, it returns an error message to the buyer.
- *PriceExplain*: Explains how PriceCompute derived a given price for a query.

QueryMarket can use any standard relational database as a back-end and uses MySQL in our implementation. It stores each dataset as a relation. Further, it also stores all the pricing information as relations. This allows us to adapt standard libraries and tools [6, 10, 11] to visualize and update prices.

QueryMarket uses various heuristics to quickly compute the prices of many simple types of queries and for common types of pricing functions such as pricing all answers tuples the same (as seen in Microsoft Azure Marketplace). Given arbitrary prices for selections on the various columns of the datasets, QueryMarket can price the class of queries called "Generalized Chain Queries". These queries were formally defined in prior work [8], we only give an intuition. Generalized Chain Queries are full conjunctive queries whose relations can be ordered in a sequence such that for any partition into a prefix and a suffix, the two sets of relations share at most one variable; these queries include all path joins, star joins, and combination.

To price the Generalized Chain Queries, QueryMarket generates a flow graph and solves the maximum flow problem, as described in our recent work [8], using the maximum flow solver present in the Boost C++ library [4]. In case the query is not a Generalized Chain Query, we reduce the problem to an Integer Linear Program and solve it using the GLPK solver[5].

6. CONCLUSION

We demonstrate QueryMarket, a prototype data marketplace that provides an online service to buy and sell structured data using the notion of "query-based pricing". We show how buyers can ask for ad-hoc queries and get them for the cheapest possible price, how QueryMarket helps sellers specify consistent prices to their datasets, and how QueryMarket explains to buyers how their queries were priced. We demonstrate the expressiveness, ease-of-use, and effectiveness of QueryMarket on use-cases presented on real-world datasets, and compare QueryMarket with existing commercial data marketplaces.

Acknowledgments. This work is supported in part by the NSF and Microsoft through NSF grant CCF-1047815 and also grant IIS-0915054.

7. REFERENCES

[1] http://gnip.com.
[2] http://www.aggdata.com/.
[3] https://datamarket.azure.com/.
[4] http://www.boost.org.
[5] http://www.gnu.org/software/glpk/.
[6] http://code.google.com/apis/chart/interactive/docs/reference.html.
[7] http://www.infochimps.com/.
[8] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Query-based data pricing. In *Proceedings of the 31st symposium on Principles of Database Systems*, PODS '12, pages 167–178, New York, NY, USA, 2012. ACM.
[9] https://datamarket.azure.com/dataset/1899a118-d202-492c-aa16-ba21c33c06cb%.
[10] http://www.pgadmin.org/.
[11] http://www.tableausoftware.com/.
[12] http://www.wordfrequency.info.
[13] http://www.xignite.com/.