Affordable Analytics on Expensive Data

Prasang Upadhyaya
University of Washington
prasang@cs.uw.edu

Martina Unutzer
University of Washington
munutzer@cs.uw.edu

Magdalena Balazinska
University of Washington
magda@cs.uw.edu

Dan Suciu
University of Washington
suciu@cs.uw.edu

Hakan Hacigumus
NEC Labs
hakan@nec-labs.com

ABSTRACT

In this paper, we outline steps towards supporting "data analysis on a budget" when operating in a setting where data must be bought, possibly periodically. We model the problem, and explore the design choices for analytic applications as well as potentially fruitful algorithmic techniques to reduce the cost of acquiring data. Simulations suggest that an order of magnitude improvements are possible.

Categories and Subject Descriptors

H.1.0 [**Models and Principles**]: General

Keywords

data markets; optimization; data costing

1. INTRODUCTION

In the last few years, software tools for data analysis (such as Fusion Tables [2] and Tableau [6]) have matured while data has become increasingly easy to obtain (*e.g.*, Windows Azure Marketplace [3]). Easy access to data and the availability of tools to analyze that data are enabling small and medium enterprises and data enthusiasts to increasingly acquire, analyze and visualize datasets [20]. Most commonly, however, such data must be purchased from vendors who specialize in curating and selling high quality datasets (*e.g.*, Twitter). The cost of data acquisition for analysis can be substantial, for example, a subset of company information from D&B [1] goes for more than a $1 per tuple (a company).

In this paper, we first explore techniques to reduce the cost of casual or exploratory data analysis on priced data. Subsequently, we estimate the likely impact of these techniques and their ease of use for an unsophisticated data analyst. This lets us identify promising techniques that merit further algorithmic and usability research.

We make the following two assumptions in order to develop our cost-reduction techniques: (a) data vendors sell inexpensive, but inferior, versions of data along with the high-quality, but often costly, version; and, (b) many real-world analyses can tolerate some degree of imperfection.

Consider the first assumption. Quandl [4] provides, for free, financial information that might be potentially stale; while Xignite [8] sells the real-time version of this data for some (not publicly disclosed) price. In general, multiple versions of a dataset may be available, and these versions may differ in accuracy, may come with additional metadata, or with different freshness, etc. This practice of selling inferior products that are derived from a common higher quality product is called *versioning*[1] [24] in the economics literature.

To motivate the second assumption, we present various plausible scenarios. First, the buyers may be executing multiple analysis, but not all may be equally important. Some, such as detecting fraudulent customer behavior, must be accurate and fast; but others, such as choosing the items to order to replenish inventory, may permit larger margins of errors and may be computed less frequently. For another scenario, consider the process of constructing a model: one may use less accurate, but cheap data, for building coarse models that are gradually refined, followed by purchasing more accurate data only when the model has high uncertainty. In another scenario, one may use cheap data to get an overview of the data so as to determine the best subset of the more accurate, but expensive, data to purchase.

Model for Analyses. We model the analyses as a set $\vec{V}$, where

$$\vec{V} = \{(V_1, \sigma_1), \cdots, (V_k, \sigma_k)\}$$

Here, V_i is a query that denotes the i^{th} analysis and each analysis V_i is associated with a minimum acceptable quality σ_i (formalized in Section 3). The views, $\vec{V}$, are defined over a database schema and can be evaluated over any *one* of the various instances of the data, defined by D^t, where

$$D^t = \{(I_1^t, \theta_1, p_1^t), \cdots, (I_n^t, \theta_n, p_n^t)\}$$

Intuitively, each instance I_i^t corresponds to a version of the database (possibly sold by different vendors) at time t, where different versions have varying quality θ_i (formalized

[1]This differs from the notion of versioning in databases that assigns versions to a sequence of data items.

in Section 2), and p_i^t is a function of time, t', that gives the cost of tuples of instance I_i^t, when accessed at time t'.

Summary of Findings. We run simulations to quantify the impact that a few simple techniques can have on the total cost of acquiring data. Our goal is to present a picture of the magnitudes of improvements that well designed data acquisition systems can achieve and we note that our techniques are not meant to be exhaustive.

1. In the simplest case (Section 3.1), when $|\vec{V}| = |D^t| = 1$, that is, when a single analysis is conducted over a dataset with a single version, a simple relaxation in the answer quality θ can lead to $3\times$ reduction in data costs. This technique can be easily used with little buyer effort.

2. In the presence of multiple data versions (Section 3.2), that is, $|\vec{V}| = 1$ and $|D^t| \geq 2$, computing the view over multiple instances and exploiting correlations can further reduce data costs by a factor of $2\times$ to $5\times$. Unlike the previous case, this technique requires additional guidance from the buyer.

3. Reusing data across multiple analyses is the third technique (Section 3.3) that is applicable when $|\vec{V}| \geq 2$. In the special case of a single version, that is, $|D^t| = 1$, the best case cost reduction is a factor of k where $k = |\vec{V}|$ is the number of analyses; and the worst case is of no reduction. This technique requires little buyer oversight.

4. Lastly, improved support for querying updates (Section 3.4), that is, querying the changes from I_i^t to I_i^{t+1}, can achieve optimal buying of updates where buyers can purchase only those updates that provably impact their analyses.

An appealing feature of these techniques is that more than one of these techniques may be applicable at a time, with each leading to independent reduction in the data cost.

2. DATA SELLERS

We model each version of the dataset at time t as an instance I^t where the version is associated with a quality metric θ_i and a price function p_i^t. APIs [23] are the most common form of providing access to data, followed by full downloads and specialized web-interfaces. Although all of these access methods are amenable to use by a data enthusiast, we will adopt an API oriented view of obtaining data for the rest of this paper for its popularity, its ability to support updates, and for the ease of combining data APIs from different sources. When purchasing data via an API, the buyer specifies parameter values to a parameterized SQL query and pays for the query results. In the D&B example [1] from the introduction, only one parameter, the company name, can be provided as input.

We first consider the quality metric, θ_i. We model the quality as a set of attribute-value pairs; for example $\theta_i =$ {`latency = 10 minutes, noise = 2, noise_type = uniform`} specifies that the data can be at most 10 minutes outdated, with a uniformly random noise added in the range $[-1, 1]$. It is not necessary that all instances share the same set of parameters θ_i. Commonly found ways of creating versions of data are: (a) Age: Older data usually costs less than more recent data, (b) Accuracy: More accurate values are usually more expensive than less precise ones, and (c) Completeness: Some sources may only provide a random sample of the tuples while others might provide the entire data.

We assume that the quality metrics of an instance remain constant over time and are made known by the sellers (that is, we do not consider the problem of learning these quality metrics), say as part of the APIs documentation. This is the common case, as seen with APIs from Xignite, Quandl, and for a majority of Twitter APIs. The `sample` API from Twitter, though, is a notable exception: its sample fraction varies over time and is difficult to estimate. We leave for future work the modeling and incorporation, in our techniques, of such time-varying quality metrics.

For pricing, we assume a pay-per-tuple pricing paradigm. This is one of two most popular pricing methods in current use [23, 21], the other being flat-rates[2]. Further, price of many datasets reduces over time and it is much cheaper to obtain historical data than the current data. Thus, we associate a pricing function, p_i^t, with each instance that also depends on when the dataset is accessed. As an example, consider a dataset whose value at creation time is constant c and is 0 subsequently. Such a pricing function may be represented as:

$$p_i^t(t') = \begin{cases} \infty & \text{if } t' < t \\ c & \text{if } t' = t \\ 0 & \text{if } t' > t \end{cases}$$

Here, the first condition of $t' < t$ makes sure that a tuple can not be accessed from a future instance. Thus, the set of all versions of a database available to a buyer at time t are given by:

$$D^t = \{(I_1^t, \theta_1, p_1^t), \cdots, (I_n^t, \theta_n, p_n^t)\}$$

Extending the pricing functions to more sophisticated functions and to incorporate other forms of pricing such as flat-rates, or arbitrage-free pricing [18] is left for future work.

3. OPTIMAL DATA ACQUISITION

Given the various data versions D^t and the set of user analysis $\vec{V}$, our goal is to maintain $\vec{V}$ over time, at the minimum cost. The problem of optimal data acquisition can be characterized using the following dimensions:

1. *Unique Data vs Data with Versions*: That is, if $|D^t| = 1$ or if $|D^t| > 1$.

2. *Individual vs Shared Analysis*: Multiple analyses on common dataset permits sharing data to reduce cost.

3. *Exact vs Approximate Analysis*: A lower quality analysis may be acceptable to the data analyst and may be generated through various means. The minimum quality of an acceptable answer for view V_i is provided through σ_i. σ may consist of the many metrics and what follows are common examples of such metrics:

   (a) *Freshness*: The freshness of a view is quantified by specifying the oldest updated tuple in the input that is acceptable, specified by τ.

[2]Pay upfront, for unlimited access to the data, for a fixed time duration.

t	0	1	2	3	4	5	6	7	8	9	10
a	$\overline{\alpha_0}$	α_1	α_2	$\overline{\alpha_3}$				α_4	α_5	$\overline{\alpha_6}$	
b	$\overline{\beta_0}$		β_1		$\overline{\beta_2}$				β_3		$\overline{\beta_4}$
	x			x			x			x	

Table 1: Table shows a series of updates to R until time $t = 10$, and the purchased tuples by the two freshness definitions with a freshness window of $\tau = 3$. Query-based freshness (shown in the last row, 'x' denotes when the purchase was made) buys the latest outstanding updates for both 'a' and 'b' for each purchase. Tuple-based freshness buys the overlined updates when they are first applied to R.

(b) *Accuracy*: Accuracy may be defined as the maximum permissible variance of the error of the values in the output.

(c) *Completeness*: It may be defined as the maximum fraction of tuples or values in the output that may be missing.

4. *Static vs Dynamic Data*: For static data, an optimal data acquisition strategy might be computed offline, but for dynamic data, an online strategy might lead to significant cost savings.

We now formalize each of settings and explore their potential to reduce costs.

3.1 Individual Analysis Over Unique Data

This is the simplest case where a single view (V, σ) must be maintained on a single database instance, which can vary over time. We denote by I^t, the instance at time t. Assume that the buyer can tolerate stale data from instances that are at most τ time units older. We explore how a tolerance for staleness can be exploited to obtain cost reductions.

If the database instance is static, that is $\forall t : I^t = I$, then a one time purchase of the data is the optimal plan. For dynamic instances, an obvious optimization is to only recompute the view when the view is accessed and the view had been computed over an instance that is more than τ time units old. But, a further *qualitative* relaxation in the notion of fairness improves significantly over this strategy. Let E^t be the view extant *seen* by the analyst at time t. Consider the following two definitions:

1. *Query-based:* E^t is generated from some database instance no older that τ units of time. That is, $\exists t' \in \{t-\tau, \cdots, t\} : E^t = V(I^{t'})$.

2. *Tuple-based:* Every tuple $x \in E^t$ is generated from some database instance in the past τ time units. That is, $\forall x \in E^t, \exists t' \in \{t-\tau, \cdots, t\} : x \in V(I^{t'})$. Note that $E^t = \emptyset$ is a trivial but valid solution. For meaningful results, we require that E^t be maximal in size.

Example. *Suppose we have a binary relation R such that $R_0 = \{(a, \alpha_0), (b, \beta_0)\}$. The view is $V =$* `SELECT ALL FROM` *R. Let $\tau = 3$. Table 1 shows the sequence of updates until $t = 10$.*

Consider the updates at $t = 1$ and $t = 2$. By both definitions, nothing needs to be updated since the data from $t = 0$ is fresh enough.

At $t = 3$, the query-based definition must purchase (a, α_3) and (b, β_1), otherwise, the view would be more than τ units older for a. But for tuple-based freshness, only (a, α_2) needs to be purchased since b's value at $t = 1$ was β_0 and is thus fresh enough.

As R's cardinality increases, it can be shown that, asymptotically, the naive strategy (that updates V after any change to D) is $3\times$ costlier than the tuple-based freshness; while the query-based freshness is $2\times$ costlier than the tuple-based freshness. $\square$

Thus, compared to the naïve strategy, a $3\times$ cost reduction can be obtained by appropriately exploiting a tolerance for stale answers. Apart from the cost savings, there are other benefits to the tuple-based definition in the context of analysis on big data:

- *Parallelize-able*: Since the decision to update any tuple is independent of the decision for the other tuples, tuple-based freshness guarantees can be provided easily in a distributed setting with no communication cost.

- *Fine-grained control*: One benefit of tuple-based definition is that it can provide different freshness guarantees for different tuples.

- *Ease of use*: This technique is very easy to use, even for a person who is not a data expert. They need to provide how stale the data can be and if tuple-based freshness is acceptable.

The primary drawback of the freshness-based definition is that the semantics of the views are more complex. This is the same kind of disadvantage that eventually consistency presents as compared to ACID.

3.2 Individual Analysis over Data with Versions

In this case, although there is a single view to maintain, the analyst has a choice of different versions of the same underlying data to choose from. For example, versions might be stale by a certain amount of time, or only have approximate values, or may have missing entries.

Intuitively, if the view's quality permits working with such approximate or stale data, the data should only be purchased from the cheapest source that provides acceptable quality. We explore some of the potential benefits of using such approximate sources through the following example. We focus on the trade-off between cost and quality.

Weather Application. Consider an application that provides the maximum temperature observed in the state of Washington in the USA. We use the historical dataset of weather metrics[3] for the state of Washington from 1890 to 2012, available from NOAA [7]. We call this the `exact` dataset.

To examine the effect of purchasing lower-quality but cheaper values instead of the exact values, we artificially create two low quality versions of `exact`: (a) `rounded`, where we round the maximum temperature to the nearest multiple of ten (for example, 33 rounded to 30); and (b) `uniform`, where a

[3]The metrics were: maximum temperature, minimum temperature, rainfall, snow, snow depth.

Dataset	Price	Variance
`actual`	12.0	0.08
`rounded`	0.122	8.17
`uniform`	0.120	8.30

Table 2: Prices of a tuple in the dataset and variance of the noise for the `actual` data and its two approximate version `rounded` and `uniform`.

uniformly random `float` from $[-5, 5]$ is added. Note that for both `uniform` and `rounded`, the true value lies within ± 5 of their respective values.

The lower-quality data is priced according to its variance as proposed in recent work [19] on pricing noisy values: the price of each tuple is $1/v$, where v is the variance of the noise[4]. Table 2 shows the prices and the variance.

As expected, purchasing values with a larger noise decreases cost (almost by a factor of 100 in our case). But the view may require temperature values with lower variance than that provided by `rounded` and `uniform`. The naïve solution is to then only buy tuples from the `exact` dataset.

In this case, however, it is possible to further reduce costs. Consider the following hybrid strategy: each day, we buy the rounded value r from `rounded` and u from `uniform`. The true value must lie in the intersection of the $r\pm5$ and $u\pm5$. If the size of the intersection is more than the accuracy threshold, ε, defined in the view's quality metric σ, we buy the exact value from `actual`. Otherwise, we return the mid-point of the intersection.

Figure 1 shows the distribution of the rounded absolute difference between the hybrid strategy outlined above and the actual values, for various values of the threshold of error, ε. We compute this distribution over each day from 1890 to 2012. As can be seen, the number of days for which the expensive exact value must be computed decreases exponentially as the ε increases. Figure 2 shows the trade-off between the standard deviation of the error of the hybrid strategy and its relative price (compared to the price of only purchasing from the `exact` dataset), for the various values of ε. As seen, even if a standard deviation of 1.0 for the error is acceptable, the prices reduce by $2\times$; and for the error's deviation of 2.0, the prices are $50\times$ lower.

Challenges. The primary challenge of using such techniques is to develop an easy way for the user, who may not be a database expert, to specify various possible ways of combining the set of data sources to obtain desired answers for cheap. Although it may be possible to automatically infer such cost-saving combinations, we know of no algorithm that can do so.

3.3 Shared Analyses Over Unique Data

Unlike the previous cases, we now have more than one view over a single shared instance of the database. That is, $|\vec{V}| \geq 2$ and $|D^t| = 1$.

The naïve solution would be to treat each view in isolation and make purchases accordingly. This can easily lead to repeat purchases. In general, it should be possible for the views to share data with each other as long as the data being shared is not too stale for a view. Note that this staleness is not a property of the data, but of the fact that a view may have purchased a data before another view requested it. If the data was purchased too long ago, such data sharing may not be possible, given the quality metrics σ_is.

Theoretically, if all views are identical, sharing data leads to a k-fold reduction in data costs, where k is the number of views. Alternatively, if the views do not access any tuples that are common, there is no benefit of this technique. In practice though, the cost savings would depend both on the amount of data shared as well as the quality requirements σ. We show an example.

EXAMPLE. *Consider relation* $I = \{(1,a),(2,b),(3,c),(4,d)\}$ *and two views* V_1 *and* V_2. V_1 *selects the first three tuples (with first attribute in the set* $\{1,2,3\}$*), while* V_2 *selects the last three tuples (with first attribute in the set* $\{2,3,4\}$*). Let*

[4]For the `exact` data purchases, we estimate the variance from the significant digits in the data: since all numbers are rounded to integers, so the noise is akin to a uniform distribution with range 1.

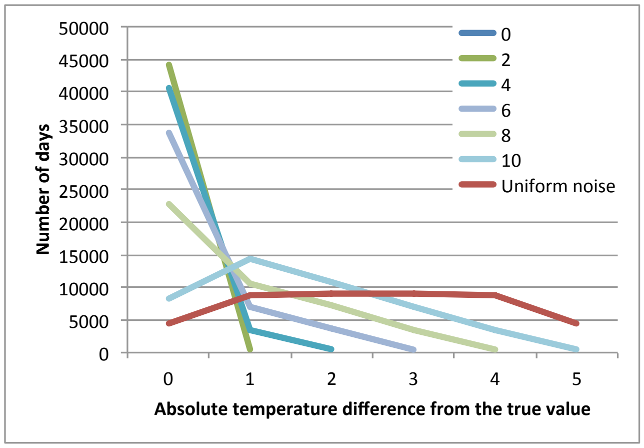

Figure 1: The distribution of the absolute difference between the approximate temperatures (from noisy sources) and the true temperatures, for various acceptable error ranges. "Uniform noise" is the distribution for the source that uniformly adds noise in the range: $[-5.0, 5.0]$.

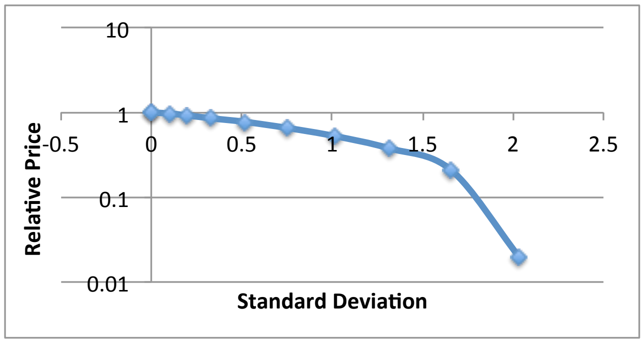

Figure 2: The observed trade-off between the average price (y-axis) of using the approximate sources (relative to buying only from the original source with no noise) and the observed standard deviation (x-axis) of the error in the approximate answers. The markers from left to right are for error ranges of $\{0, 1, \ldots, 10\}$.

the update threshold for V_1 be $\tau_1 = 2$ and for V_2 be $\tau_2 = 3$ (all in seconds and using query-based freshness).

Let all the tuples be updated every second.

If the views are updated after the end of their respective freshness thresholds, V_1 is updated every τ_1 seconds, while V_2 is updated every τ_2 seconds. Note that since the data is updated every second, V_2 can not always reuse the common tuples from the tuples purchased for V_1 otherwise V_2 would violate query-based freshness semantics. Thus, on average, this strategy purchases $\frac{3}{\tau_1} + \frac{3}{\tau_2} = 2.5$ tuples per second.

On the other hand, if both V_1 and V_2 are updated simultaneously every τ_1 seconds, the average is $\frac{4}{\tau_1} = 2$ tuples per second.

Thus, the former is $0.25\times$ costlier than the latter. $\square$

Challenges. It is easy to compute closed-form solutions for when to update each view if it is known which views use a given tuple. In the example above, if the cardinality of V_2's input shared with V_1 is n_{12} and the total cardinality of V_2 in n_2, then the second strategy is preferred when $\frac{\tau_2}{\tau_1} < \frac{n_2}{n_2 - n_{12}}$.

The problem with this solution is that the *entire* data may not be updated *every* second and the optimal schedule would depend on the exact tuples updated.

Further, sharing data with tuple-based freshness is, surprisingly, much harder to solve optimally since each tuple is effectively a separate view and thus, the number of potential overlaps to consider becomes exponential in the size of the query answer and the number of views.

3.4 Identifying Relevant Updates

The last aspect we consider is that while purchasing data that is updated, it is not required to buy all the updates. Unfortunately it is not always obvious without buying the update if it is relevant. Formally, the problem is: given a view V over instance I^t, what new tuples must be purchased from I^{t+1} to update the $V(I^t)$?

If the only way to know of updates is by purchasing the tuples themselves, there is little possibility for cost reduction apart from only buying those tuples that are in the provenance of the view[5]. Thus, to use this technique, additional support from the data seller is needed. We consider the following features:

- *Row updates*: In this case, the keys of the updated rows are freely available.
- *Cell updates*: In this case, both the keys and the columns of the updates are freely available.
- *Counting-based*: In this case, the count of any query is available for free. An example is the Salesforce ContactCount API [5]. If only one cell update is made at a time, one can keep counting the cardinality of the view's output and buy an update when the cardinality changes.

We explain through an example.

EXAMPLE. *Let $I^0 = \{(1, \alpha_1, \beta_1), (2, \alpha_2, \beta_2)\}$ be an instance of relation R at time $t = 0$ and the view V be the result of the query $Q(x)$:- $R(x, y, z), y = \alpha_1$. Here the first attribute is the key. In our notation, $V(R)$ denotes the evaluation of the view on database R.*

If an update changes β_1 to β_1' in tuple 1, $V(I^0)$ changes and hence, the update must be purchased.

If we only have row updates, we can only know that tuple 1 was updated, we must buy the update and check. But with cell updates, we can infer that we need not buy the update since the change is to a column that is not relevant for V.

Of course, not all vendors may support querying updates one-at-a-time. Consider now two updates and one insertion to I^0 that collectively swap the value of the second attribute between the two tuples and adds a third unrelated tuple. Now, $I^1 = \{(1, \alpha_2, \beta_1), (2, \alpha_1, \beta_2), (3, \alpha_3, \beta_2)\}$. $V(R)$ has changed but its cardinality has not and it is impossible to infer the meaningful updates now through any of the techniques in isolation or together. $\square$

As seen from the example, batched updates are significantly more challenging to analyze for inferring the updates to purchase. But individual processing of updates can add a significant performance penalty.

Updates and Freshness. Of course, if the view can tolerate stale data, that is, updates must only be purchased every τ time units, for certain classes of views, batch updates may be the preferred purchasing method. We briefly analyze the following cases. Here, *batch* updates refers to buying the updates only from instances at every τ time units and *serial updates* refers to those that may buy tuples from intermediate instances too.

1. *Full queries with selections*: Consider queries of the form:

   $$Q(x, y) \text{ :- } R(x, y), C(x)$$

   Here, C is a predicate that performs a selection on attribute A. With row updates, every update must be purchased unless $C(x)$ does not exist (no selections).

   With column updates, if the update is to column A, it must be bought; if the update is to B and the corresponding value of A, say a, is such that $C(a)$ was `true` and hence was part of the computed view, then also the update must be bought; else the update can be discarded. Here too, collapsing the updates (*i.e.*, batching) outperforms serial update purchases.

   The counting-based approach can be applied only in an update-at-a-time case. In such a case, it can avoid buying new tuples that evaluate to false to the predicate C, but it may end up purchasing additional updates if updates overwrite a previously updated tuple.

2. *Projections with selections*: Consider queries of the form:

   $$Q(y) \text{ :- } R(x, y), C(x)$$

   In the bag semantics, this is equivalent to full queries with selections. Thus we consider set semantics. As shown in the example previously, any update to either of A or B must be purchased if row updates or cell updates are used.

   With update-at-a-time and the counting-based approach, we can only purchase the tuples that are relevant.

[5]We must buy all newly inserted tuple the first time they are inserted.

Challenges. Incorporating a tolerance for approximations can be a potentially fruitful technique. One way to do so is to use models for updates to compute approximate answers. For instance, the updates to the current temperature change slowly on an hourly basis. This information can be used to avoid doing additional data purchases.

4. RELATED WORK

The techniques outlined in this paper touch upon many areas of database research and we summarize some of the important connections.

Freshness. The notion of freshness (or currency) has been explored in the context of querying asynchronously maintained replicas in distributed and replicated databases [13, 16], using query-based freshness. Supporting finer-grained freshness and optimizing for prices as opposed to performance is a significant difference in goals and assumptions.

Approximate Query Answering. It is an active area of research, deeply explored in the context of streaming data [12, 9], sensor data [15], and approximate answers on relational data [17, 10]. We expect that such techniques should be directly usable in the context of the problem in this paper.

View Maintenance. Prior work has focused on computationally efficient incremental view maintenance techniques [14, 11], given the updates. The main constraint in this paper's case is that the we must determine if an update is relevant for a view with restrictions on what part of the update is known, and neither class of research answers this.

5. DISCUSSION

We envision that the eventual goal of this line of work is to support data analysis given a budget for data acquisition. This paper takes one step to the realization of such systems, by outlining the opportunities for cost reduction in the setting of shared data analyses, and given the constraints of the APIs that current data providers employ.

Future work falls into two categories. First, buyer-side optimizations and buyer-side estimation of goodness of sources [22] is likely to drive significant improvement in the quality of data analytics while containing costs in a predictable manner (say a fixed budget).

The other direction of work is improvements in data provider APIs. There is significant experimentation [23, 21] underway in the markets comprising of data sellers. It would be fruitful to research the design of better data APIs (for example, APIs that allow for cheap determination of updates)jsto ease customizations and optimizations for the buyers, but that are still efficient for the sellers to support.

6. CONCLUSION

We formalize the problem of optimal data acquisition in a setting where the data analyst must purchase the data from vendors who sell data for profit. We outline, and explore through simulations, various promising design choices and techniques that can provide an order of magnitude reduction in costs. We find that the most promising avenue for cost reductions is the use of approximate answers and allowing for fine-grained freshness of answers. Smaller savings in data costs are obtained with optimizing for updates and sharing data across multiple views.

7. ACKNOWLEDGMENT

This work is supported in part by NSF grant IIS-1115188 and a gift from NEC Labs.

8. REFERENCES

[1] Company Firmographics Industry and Sales. http://datamarket.azure.com/dataset/dnb/companyfirmographics.

[2] Google Fusion Tables. tables.googlelabs.com/.

[3] Microsoft Azure Marketplace. http://datamarket.azure.com/.

[4] Quandl. http://www.quandl.com/help/api.

[5] Salesforce ContactCount API. https://www.data.com/export/sites/data/common/assets/pdf/DS_Datadotcom_Connect_API_Docs.pdf.

[6] Tableau. http://www.tableausoftware.com/.

[7] United States Historical Climatology Network (USHCN) Daily Dataset. http://cdiac.ornl.gov/ftp/ushcn_daily/.

[8] Xignite. http://www.xignite.com/Products/.

[9] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. Aurora: A new model and architecture for data stream management. *The VLDB Journal*, 12(2):120–139, Aug. 2003.

[10] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 29–42, New York, NY, USA, 2013. ACM.

[11] Y. Ahmad and C. Koch. Dbtoaster: A sql compiler for high-performance delta processing in main-memory databases. *Proc. VLDB Endow.*, 2(2):1566–1569, Aug. 2009.

[12] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '02, pages 1–16, New York, NY, USA, 2002. ACM.

[13] P. A. Bernstein, A. Fekete, H. Guo, R. Ramakrishnan, and P. Tamma. Relaxed-currency serializability for middle-tier caching and replication. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 599–610, New York, NY, USA, 2006. ACM.

[14] L. S. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey. Algorithms for deferred view maintenance. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, SIGMOD '96, pages 469–480, New York, NY, USA, 1996. ACM.

[15] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. VLDB '04, pages 588–599, 2004.

[16] H. Guo, P.-Å. Larson, R. Ramakrishnan, and J. Goldstein. Relaxed currency and consistency: How to say "good enough" in sql. In *SIGMOD Conference*, pages 815–826, 2004.

[17] C. Jermaine, S. Arumugam, A. Pol, and A. Dobra. Scalable approximate query processing with the dbo engine. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, SIGMOD '07, pages 725–736, New York, NY, USA, 2007. ACM.

[18] P. Koutris, P. Upadhyaya, M. Balazinska, B. Howe, and D. Suciu. Query-based data pricing. In *PODS*, pages 167–178, 2012.

[19] C. Li, D. Y. Li, G. Miklau, and D. Suciu. A theory of pricing private data. In *ICDT*, pages 33–44, 2013.

[20] K. Morton, M. Balazinska, D. Grossman, R. Kosara, and J. Mackinlay. Public data and visualizations: How are many eyes and tableau public used for collaborative analytics? In *SIGMOD Record*, 2014.

[21] A. Muschalle, F. Stahl, A. Löser, and G. Vossen. Pricing approaches for data markets. In *BIRTE*, pages 129–144, 2012.

[22] T. Rekatsinas, X. L. Dong, and D. Srivastava. Characterizing and selecting fresh data sources. In *SIGMOD Conference*, pages 919–930, 2014.

[23] F. Schomm, F. Stahl, and G. Vossen. The data marketplace survey revisited. Technical report, ERCIS European Research Center for Information Systems, 2014.

[24] H. Varian. Markets for information goods. Technical report, University of California, Berkeley, 1998.