Caravan: Provisioning for What-If Analysis

Daniel Deutch
Ben-Gurion Univ.
daniel.deutch1@gmail.com

Zachary Ives
Univ. of Pennsylvania
zives@cis.upenn.edu

Tova Milo
Tel Aviv Univ.
milo@post.tau.ac.il

Val Tannen
Univ. of Pennsylvania & EPFL
val@cis.upenn.edu

ABSTRACT

Problems of *what-if analysis* (such as hypothetical deletions, insertions, and modifications) over complex analysis queries are increasingly commonplace, e.g., in forming a business strategy or looking for causal relationships in science. Here, data analysts are typically interested only in task-specific views of the data, and they expect to be able to interactively manipulate the data in a natural and seamless way — possibly on a phone or tablet, and possibly via a spreadsheet or similar interface without having to carry the full machinery of a DBMS.

The Caravan system enables what-if analysis: fast, lightweight, interactive exploration of alternative answers, within views computed over large-scale distributed data sources. Our novel approach is based on creating dedicated *provisioned autonomous representations*, or PARs. PARs are compiled out of the data, initial analysis queries and user-specified what-if scenarios. They allow rapid evaluation of what-if scenarios without accessing the original data or performing complex query operations. Importantly, the size of PARs is governed by the parameters of the what-if analysis and is proportional to the size of the initial query *answer* rather than the typically much larger source data. Consequently, many what-if analysis tasks performed through PAR evaluations can be done autonomously, on limited-resource devices. We describe our model and architecture, demonstrate preliminary performance results, and present several open implementation and optimization issues.

1. INTRODUCTION

With the advent of large scientific community repositories, business-to-business cooperatives, and enterprise warehouses — there are increasingly many settings with large numbers of relations and interrelated derived views, whether within a single database instance [9] or across multiple distributed sites (e.g., data exchange [6] or collaborative data sharing [14]). Such sources of data may be used to answer a vast array of different questions. However, a typical end user of the data will likely be interested in a fairly restricted set of information: e.g., an account executive may care about the data and customers within his or her region, or a scientist may care about specific organisms or genes. Ideally, the user will interact with the relevant data via an interactive application — whether a domain-specific application used by the analyst, a spreadsheet interface over a DBMS [16, 18], a visualization tool, or even a conventional spreadsheet. Today, such a spreadsheet might not even be hosted on a conventional PC, but rather a tablet or a phone!

The end user's goal may be to understand the data in his or her interest area, and to experiment with predictions, forecasts, and diagnoses. Hence he or she will often want to do more than pose single static queries: he or she may want to **interact** with the system by drilling down or *refining a query*, or by considering *hypothetical updates* — modifying some assumptions, projecting an outcome, postulating a connection, etc. In general, the goal is to perform some form of *what-if analysis*, over a view or set of related views over the data — without having to modify the original copy of the data and producing side effects elsewhere within the data management infrastructure. We illustrate with a simple running example.

EXAMPLE 1.1. *An analyst working for PhoneCo is looking at data involving customers, their current subscriptions to calling plans, plan prices, and call durations, using the following schema:*

Cust(Num,Name,State) Subscr(Num,Plan)
Plan(Name,Mo,Price) Call(Num,Mo,Dur)
MonthQuarter(Month,Quarter)

The call durations in minutes are already totaled by month for each customer. Plan per minute prices may vary from month to month. Now consider an analysis question, "Compute the company revenues grouped by and totaled by calling plan, quarter, and month." Perhaps the user's original view of the data might be captured in SQL as:

```
SELECT Plan, Quarter, C.Mo, SUM(Dur * Price)
FROM   Call C, Subscr S, Plan P, MonthQuarter MQ
WHERE C.Num=S.Num AND Plan=P.Name AND
        C.Mo=P.Mo AND MQ.Month = P.Mo
GROUP BY ROLLUP (Plan, Quarter, C.Mo)
```

The user, performing as an analyst, may want to explore the revenue figures under some "what-if" scenarios:

S1. What if prices for Connecticut and Pennsylvania customers during some *quarter are decreased by 20%?*

S2. What if Connecticut customers made no calls in January?

S3. What if the analyst wants to see just the revenue from customers in Pennsylvania that subscribe to Plan A*?*

All these scenarios involve query refinement, and specifically they may require additional detail about State *data. S1 asks for hypothetical modifications while S2 and S3 ask for hypothetical deletions. Note also that S1 has a parameter that can be set to any of 4 quarters. (The percentage could also be a parameter, one that takes continuously many values.) We can also think that each scenario is associated with a Boolean parameter which encodes that the scenario is on/off. All in all, there are* $5 \times 2 \times 2 = 20$ *(counting the Boolean conditions and parameters for each scenario) separate*

CIDR'13 6th Biennial Conference on Innovative Data Systems Research, January 6-9, 2013, Asilomar, California, USA.
.

combinations of parameter values that yield (in general) 20 *distinct answers, and the analyst may want to explore any number of them.*

The previous example also emphasizes an important challenge: an analyist may formulate a reasonably short list of what-if scenarios, like the three above, but will in principle be interested in any of the possible ways of considering subsets of these scenarios (turning any of them "on/off') and of all possible values given to the various parameters. Throughout the paper we will refer to these many more situations as *combinations of scenarios* keeping in mind that the number of such combinations is exponential in the number of scenarios/parameters.

The question, then, is how to enable the user to *interact* with a view instance by refining such parameters, e.g., as the ones associated with our three scenarios in the above example. Ideally, each time the user tweaks parameters and tests a combination of scenarios, the system should recompute the contents of the view at interactive rates (generally considered to be 300msec or faster). Many spreadsheets can provide such responsiveness, but in general spreadsheets operate in main memory and do not scale to large OLAP settings. Conversely, what-if analyses are only addressed in a limited way using today's widely available OLAP technologies — which help with certain query refinements but not with hypothetical updates, since they are focused on querying over existing state. (Some past investigations of querying under hypothetical updates, e.g. [8, 12, 4], are discussed in Section 6.) To accomplish what-if analysis today, we must have direct access to the source data or a replica, and must determine the effects of query refinement and hypothetical updates by reformulating the initial query and maybe even performing some transactional updates. Moreover, we must repeat this for each combination of scenarios of interest. Such repeated computations can get expensive as the queries are posed over multiple levels of views and aggregates, or in a distributed setting. They are unlikely to provide sub-second response rates, and moreover, they rely on having direct access to full DBMS machinery and all of the data underlying the view(s).

Intuitively, it should be possible to *precompute* some parameterized data instance (where the parameters come from the what-if scenarios) which then enables rapid and autonomous recomputation of the effects of updates over the view, analogous to the way today's DBMSs use materialized views to speed up evaluation of complex queries. This parameterized data instance would have to maintain the information that is relevant to scenarios, information that would otherwise not appear in the original view (e.g., in Example 1.1, to deal with the discounting in scenario S1 it would have to preserve separately the computed revenue from Pennsylvania and Connecticut for each quarter). Moreover, it would have to do this for *all* what-if scenarios at once and only preserve the "minimum necessary" information such that we do not preserve, e.g., the entire source database. We refer to the problem of creating such parameterized instances as **provisioning** and to the instances as **provisioned autonomous representations** or PARs.

Provisioning introduces many of the same problems as view materialization: choosing what to materialize, trading off space and time, and using the PAR to compute results for a combination of what-if scenarios. While the example above is simple for expository reasons, in general we expect typical what-if scenarios to be similarly characterized by a fixed and relatively small number of parameters or options. We show that the corresponding PARs are in general much more compact than materialized views, and may be created for highly complex OLAP views. Moreover, we consider it highly desirable that the PAR can be stored in a spreadsheet or similar client application, and that the view instance corresponding to a combination of scenarios can be rapidly evaluated over the PAR in a single pass. Therefore the size of the PAR should be comparable to the size of (the largest of) the answers/views that it represents, rather than the size of the source data, or even of a significant portion thereof. This introduces a number of issues in determining how to most efficiently compute and encode the data for the PAR, as required for the different combinations: how aggressively do we combine common subexpressions, when should we add rows versus columns to capture different scenario combinations, etc. Of course, as we encode the PAR we must additionally generate the view, spreadsheet, or user interface code that evaluates scenario combinations over it. Our work on the Caravan system has begun an exploration of these issues.

The rest of this short paper is organized as follows. Section 2 presents an overview of our system architecture. In Section 3 we describe the formal model for provisioning, and in Section 4 we show how to compute PARs. We next present preliminary results to illustrate the three-orders-of-magnitude and higher performance gains provided by PARs. Finally, we discuss related work in Section 6 with conclusions and future work in Section 7.

2. CARAVAN SYSTEM ARCHITECTURE

As illustrated in Figure 1 provisioning is achieved via two components (separated by the vertical dashed in the figure) with quite different computing characteristics. In the first stage, we have the Caravan system, a "thin" middleware layer that wraps with additional capabilities an existing query system (in the current Caravan implementation this is IBM DB2 or MySQL, but it could alternatively be an enterprise information integration system). Caravan passes through standard DBMS requests, but it is especially designed to support what-if analysis over OLAP queries with lightweight clients. Given the definition of a view of interest to the user, as well as a set of possible what-if scenarios (both defined in a novel *provisioning query language* that extends SQL), Caravan *provisions* by computing one or more PARs. Intuitively, PARs represent minimal, compressed, parameterized instances from which answer instances under any combinations of scenarios can be computed. The requirements of this first stage are similar to those needed to compute the view itself, whether a server-based RDBMS or a query system over federated databases.

PAR(s) are then transferred to the second component, an interactive client application (currently a spreadsheet or a lightweight DBMS). This second stage's requirements are much simpler. The client application does not present PARs directly to the user, but rather evaluates them for different parameter values, yielding a single database view instance. A useful metaphor (continued in Section 3) is that of a "dashboard" with "switches", "knobs", and a "go" button. Giving values to the parameters that appear in a PAR corresponds to the user setting switches and turning knobs. The user then hits the go button and the application displays the view instance that corresponds to the chosen switch/knob settings. This can be repeated indefinitely, and does not affect the original database unless the user specifically requests updates in accordance with the knobs' settings. In our current platform, we have implemented such a dashboard both with a spreadsheet, in which case the data seen by the user is a tab that contains formulas linked to the PAR; and with a lightweight DBMS, in which case the data seen by the user is a view computed based over the PAR instance.

3. PROVISIONED AUTONOMOUS REPRESENTATIONS

In this section we describe the main ingredient of our approach to provisioning. We intend *provisioned autonomous representa-*

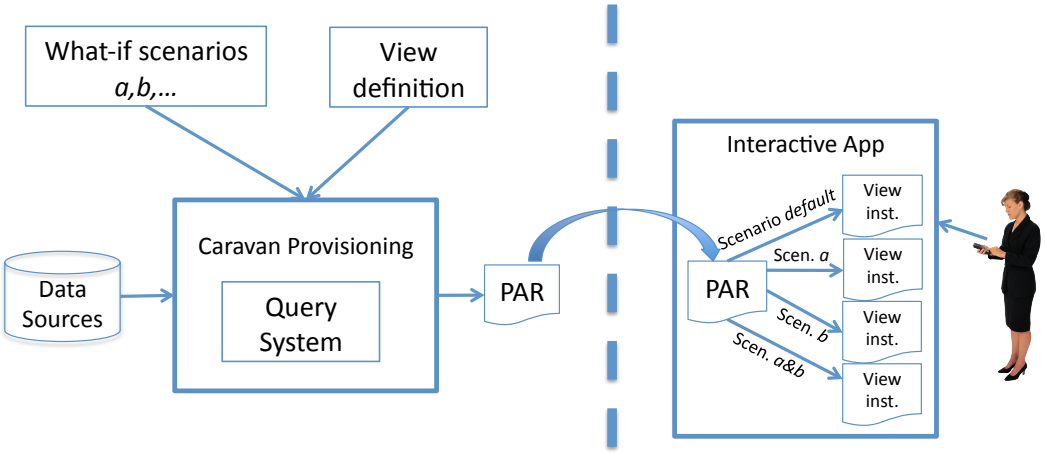

Figure 1: Caravan layers over a query system to compute provisioned autonomous representations (PARs) from view definitions and lists of what-if scenarios. PARs are then transfered to a client spreadsheet or other interactive application. Based on user interaction different combinations of scenarios can be selected, returning different instances of the view.

tions (PARs) to have the ability to produce in a lightweight fashion the answer to any combination of the what-if scenarios from which they were built. Since we need compact single representations of multiple answer instances, it is natural to appeal to past work on incomplete databases and specifically to the *conditional tables* (c-tables) of [13]. In c-tables, the tuples are associated (annotated) with *conditions*: Boolean expressions involving variables (parameters). When the parameters are given values, these annotations evaluate to true of false. Keeping only the tuples annotated true yields a conventional relation.

For us, the parameters correspond to the what-if scenarios. Continuing the dashboard metaphor introduced in Section 2 we think of giving values to the parameters as setting switches and turning knobs. Typically, for each scenario we have a Boolean parameter (a switch parameter) which is valuated to true when the scenario is on and false when it is off. Knob parameters capture additional aspects of the scenarios and can be string, integer, or real values as needed. Boolean parameters participate directly in the annotations while we use the Boolean-valued construct $[\texttt{Attr}\ \theta\ p]$ to involve the knob parameter p in annotations (θ is equality, inequality, etc.).

C-tables are not appropriate for queries with aggregation (they lead to an exponential blow-up in data complexity [3]). Following [3] we associate (annotate) also the values that participate in an aggregation with conditions—Boolean expressions. Suppose we sum-aggregate n values of Attr such that the tuple in which v_k appears is annotated with the (condition) Boolean expression b_k, $k = 1, \ldots, n$. We express the result as

$$(v_1)b_1 + \cdots + (v_n)b_n$$

and we allow such expressions to appear in the fields of a PAR. The use of $+$ in the expression above is akin to the use we make of operation symbols in conventional algebraic expression: while we have variables, we can only do algebraic manipulations; however when the variables are replaced by numbers then we can do arithmetic and get a numerical result. Indeed, the key to dealing with annotated aggregation expressions efficiently, are their rich algebraic properties [3], specifically the fact that + is associative and commutative and

$$\begin{array}{rcl@{\qquad}rcl} (u)b + (v)b & = & (u+v)b & (v)\mathsf{false} & = & 0 \\ ((v)b_1)b_2 & = & (v)b_1b_2 & (v)\mathsf{true} & = & v \end{array}$$

where b_1b_2 is short for $b_1 \wedge b_2$.

EXAMPLE 3.1. *Continuing example 1.1 we show in Figure 2 (a) the PAR generated from the scenarios S1,S2,S3, from the original revenue view definition shown in example 1.1 and from some sample data, a PhoneCo instance similar to the one used in the experiments described in Section 5. In the sample data, which assumes two calling plans, the tables* Cust *and* Subscr *have on the order of* 400K *tuples each and* Call *has on the order of* 4.8M *tuples assuming that it totals durations for each of* 12 *months.*

Therefore the PAR is a table with $2 \times (12 + 4 + 1) = 34$ *tuples (only 5 rows are shown in Figure 2 for simplicity) featuring switch (Boolean) parameters* s_1, s_2, s_3 *where* $s_k =$ true *iff Sk is on, and the additional knob parameter* p_1 *(used in S1) that can be set to any of* 1st, 2nd, 3rd, 4th. *Combinations of scenarios correspond to assignments of values for these parameters. Notice that the PAR has both annotated values (in the sum-aggregations) and annotated tuples and that the annotations are boolean expressions. The example can be understood just from the following evaluation rules: (1) tuples annotated with* true *are kept, (2) those annotated with* false *are deleted, (3)* $(v)\mathsf{true} = v$, *and (4)* $(v)\mathsf{false} = 0$. *Notice that the* Amt *fields contain aggregations of values associated (annotated) with Boolean expressions (when the annotation is just* true *it is omitted since* $(v)\mathsf{true} = v$*). The aggregations that appear in the PAR involve only a small number of terms (mostly up to 4 terms but in one global summary row we have 12) even though they result from the grouping-aggregation of millions of tuples. This is because the computation of the PAR takes advantage of the algebraic laws that govern these expressions: distributivity of annotation over sum, associativity, commutativity, etc. The number of terms in the PAR aggregation expressions is not determined by the size of the input data, but by how many separate combinations of scenarios we must keep track of. The result is a small PAR that yields, at interactive speeds, any of the conventional answers of interest — in particular those shown in Figures 2 (b), (c), and (d). Note, for example, that under S3 the annotation of the* B-

Plan	Qrt	Mo	Amt	*cond*
A	1st	Jan	$(750\text{K}) + (-150\text{K})s_1[p_1 = \texttt{1st}] + (500\text{K})\overline{s_2}\,\overline{s_3} + (-100\text{K})s_1[p_1 = \texttt{1st}]\overline{s_2}\,\overline{s_3}$	true
B	1st	Feb	$(600\text{K})\overline{s_3} + (-40\text{K})s_1[p_1 = \texttt{1st}]\overline{s_3}$	$\overline{s_3}$
A	1st	null	$(1800\text{K}) + (-360\text{K})s_1[p_1 = \texttt{1st}] + (500\text{K})\overline{s_2}\,\overline{s_3}+$	
			$+(-100\text{K})s_1[p_1 = \texttt{1st}]\overline{s_2}\,\overline{s_3} + (650\text{K})\overline{s_3} + (-130\text{K})s_1[p_1 = \texttt{1st}]\overline{s_3}$	true
B	2nd	null	$(2600\text{K})\overline{s_3} + (-240\text{K})s_1[p_1 = \texttt{2nd}]\overline{s_3}$	$\overline{s_3}$
B	null	null	$(12000\text{K})\overline{s_3} + (-360\text{K})s_1[p_1 = \texttt{1st}]\overline{s_3} + (-240\text{K})s_1[p_1 = \texttt{2nd}]\overline{s_3}$	
			$+(-360\text{K})s_1[p_1 = \texttt{3rd}]\overline{s_3} + (-240\text{K})s_1[p_1 = \texttt{4th}]\overline{s_3}$	$\overline{s_3}$
...	...	...	...	...

(a)

Plan	Qrt	Mo	Amt
A	1st	Jan	1250K
B	1st	Feb	600K
A	1st	null	2950K
B	2nd	null	2600K
B	null	null	12000K
...	...	...	...

(b)

Plan	Qrt	Mo	Amt
A	1st	Jan	600K
A	1st	null	1440K
...	...	...	...

(c)

Plan	Qrt	Mo	Amt
A	1st	Jan	600K
B	1st	Feb	560K
A	1st	null	1960K
B	2nd	null	2600K
B	null	null	11640K
...	...	...	...

(d)

Figure 2: **(a)** Some rows from the provisioned autonomous representation (PAR) for S1, S2, and S3; **(b)** answer to initial query, all scenarios off: $s_1 = s_2 = s_3 = \mathsf{false}$; **(c)** all scenarios are on (S1 with first quarter): $s_1 = s_2 = s_3 = \mathsf{true}, p_1 = \texttt{1st}$; **(d)** S1 and S2 are on (S1 with first quarter) but S3 is off: $s_1 = s_2 = \mathsf{true}, p_1 = \texttt{1st}, s_3 = \mathsf{false}$.

tuples of the PAR is $\overline{s_3} = \mathsf{false}$ *so they go away in Figure 2 (c). Note also that under S1 with the first quarter knob choice we have* $[p_1 = \texttt{2nd}] = \mathsf{false}$ *so revenue in the second quarter does not differ between Figures 2 (d) and (b).*

Our example illustrates a quite general situation: the size of (annotated) result depends primarily on the number of scenarios, rather than on the source data size. This observation is key for the applicability of provisioning.

In a similar fashion to sum-aggregation, our framework also supports max/min-aggregation (and count and average are derived from sum). The entire framework is compositional with respect to query constructs and allows for nested aggregation, since we can put aggregation expressions everywhere where a value is expected, even within comparison predicates in condition annotations.

The Boolean-valued construct $[\texttt{Attr}\ \theta\ p]$ is a particularly flexible device. For example, if our view definition also included

HAVING Amt > 1300K

then our PAR would include conditions such as

$$\overline{s_3}[(1400\text{K})\overline{s_3} + (-160\text{K})s_1[p_1 = \texttt{1st}]\overline{s_3}\ >\ 1300\text{K}]$$

In the end, given a parameter valuation all such conditions still evaluate to true or false.

4. IMPLEMENTING PARS

PAR implementation is comprised of three interrelated aspects. The first is computing the PAR efficiently given the original view definition, the set of scenarios and a (possibly distributed) data instance. The second aspect is finding compact encodings for PARs and the third is fast evaluation of PARs producing view instances that correspond to user-specified choices of scenario combinations. We assume data analytics-scale computing power to be available for the first aspect, but only limited-resource environments for the second and third.

4.1 Provisioning the View

In a first phase, Caravan takes a view definition and a set of what-if scenarios and *rewrites* the view definition into a *provisioning computation* according to rules that are compiled by Caravan from each of the what-if scenarios (formulated in a dedicated specification language). We can think of the provisioning computation as a query computation. Recall however that PARs go beyond conventional instances in several ways: they include boolean-valued expressions built from parameters (variables), they include annotations (associations) of tuples by such expressions as well as similar annotations of values that participate in an aggregation. Neither variables nor annotations will appear in the data sources so they must be introduced by the provisioning query, which means that it will have to feature constructs beyond the standard SQL. These constructs manipulate will manipulate the scenario parameters and will build annotations. We sketch the process of our preliminary implementation here, starting with an illustrative example.

EXAMPLE 4.1. *Again continuing Example 1.1 we show how to rewrite the original view definition in order to provision for S1.*

```
SELECT Plan, Quarter, C.Mo,
    SUM(Dur * Price *
      (1 -
      (0.2) CASE WHEN State='CT' OR State='PA'
          THEN s1 AND [p1=Quarter] ELSE FALSE END
      ))
  FROM Call C, Subscr S, Plan P, MonthQuarter MQ, Cust Cu
  WHERE C.Num=S.Num AND Plan=Name AND C.Mo=P.Mo AND
      Month=P.Mo AND S.Num=Cu.Num
  GROUP BY ROLLUP (Plan, Quarter, C.Mo)
```

Note the (nonstandard SQL) construct that annotates `(0.2)` *by Boolean expressions. In order to compute a small PAR with this query the Caravan engine takes advantage of algebraic manipulations to arrange the term such that what is summed is the annotated aggregation expression:*

$$(\texttt{Dur} * \texttt{Price})\texttt{true} + (-\texttt{Dur} * \texttt{Price} * 0.2)s_1[p1 = \texttt{Quarter}]$$

During the computation of the PAR Caravan creates, for each group, two buckets, one corresponding to the annotation true *the other to* $s_1[p1 = \texttt{Quarter}]$ *and sums in each bucket the corresponding values. The resulting* `Amt` *in the PAR (for plan* `A`*,* `1st` *quarter and* `Jan`*) is* $(1250\text{K})\texttt{true} + (-250\text{K})s_1[p1 = \texttt{Quarter}]$ *This corresponds to the informal understanding of the problem:* 1250K *is the revenue when S1 is off; when S1 is on and it discounts the first quarter, we subtract* 250K *yielding* 1000K.

For scenario S2 (respectively S3), the rewriting will introduce further (semi)joins between Call *(respectively* Subscr*) and* Cust *as well as tuple annotations involving the switch parameter* s_2 *(respectively* s_3*) to capture the hypothetical deletions.*

We take the query for building the PAR and create two complementary views:

1. An SQL materialized view for the PAR itself that computes, in a separate column, each intermediate value that is conditionally used in a scenario. We discuss the contents of this view in the Section 4.2.

2. A means of computing a single instance from the PAR (via, e.g., a parameterized SQL query or even a spreadsheet). This takes parameters and the materialized view representing the PAR, and returns the results corresponding to the scenario combination. We discuss this second construct in Section 4.3.

We now describe how we achieve these two subtasks.

4.2 Encoding PARs in Relations

While standard SQL does not support direct encoding of annotations and expressions within tables, there is a reasonably straightforward mapping from a PAR to a materialized encoding. This involves involves taking each WHERE or SELECT expression, and determining the set of parameters involved. In our example, these are s1 and p1.

We next do case-based reasoning about the subexpressions dependent on each such parameter and scenario. We look at the PAR generation query and each component that is dependent on a scenario: here, s1 affects the sum for tuples where the state is CT or PA, and p1 affects tuples where the quarter is equal to the value of p1. Our initial implementation creates a separate column for each such subexpression.

EXAMPLE 4.2. *We would transform the extended SQL for scenario S1 from our previous example into the following, by observing the portion of the SUM conditioned on scenario S1. The resulting view separately computes attributes for the overall monthly revenue and the potential discount for Pennsylvania and Connecticut. We will get one attribute for each derived result used in a scenario.*

```
CREATE MATERIALIZED VIEW Scenario1_PAR_Encoding
  SELECT Plan, Quarter, C.Mo,
    SUM( Dur * Price ) AS Amt1,
    SUM(CASE WHEN State = 'CT' OR State = 'PA'
              THEN 0.2*Dur*Price ELSE 0 END) AS Amt2
  FROM Call C, Subscr S, Plan P, MonthQuarter MQ, Cust Cu
  WHERE C.Num=S.Num AND Plan=Name AND C.Mo=P.Mo AND
    Month=P.Mo AND S.Num=Cu.Num
  GROUP BY Plan, Quarter, C.Mo
```

Observe that this materialized view separates out the portion of the arithmetic expression attached to the CASE WHEN State = 'CT' OR State = 'PA' *scenario test. The test for the value of* s1 *can only be evaluated at runtime, hence this expression is given its own column. Similarly, we cannot evaluate* p1=Quarter *until runtime, and the view must output* Quarter *(which in fact it already did according to the original PAR definition).*

The above expression may need to be further factored into a query and subquery for some DBMSs, e.g., the above works in MySQL but not DB2. This is due to restrictions on how aggegation and conditionals can be applied. For DB2 we generate separate subqueries to compute the different sets of tuples to be aggregated under different scenarios, e.g., PA *vs.* CT *vs. the remaining states.*

Our preliminary implementation is relatively naive in several ways. It enumerates scenarios individually and does not consider, e.g., ways of pre-evaluating results based on scenario combinations, or ways of reusing columns if certain combinations are mutually exclusive. Moreover, for some settings it might be more efficient to encode some of the scenario data in additional *rows* as opposed to *columns*. We hope to explore such opportunities in the future.

4.3 Computing Instances for Scenarios

For the implementation that uses a lightweight DBMS, along with creating a relation encoding the PAR, Caravan simultaneously creates a parameterized view that can be used to evaluate the instance under the different scenarios. Given the rewritten view definition from Section 4.1, we further rewrite it to make use of the materialized view representing the PAR: this is straightforward, as the materialized view precisely matches it except that it separately encodes the intermediate results for operations that are conditioned on scenarios or parameters. Each subexpression that is conditioned by a variable is in its own column in the materialized view, and we can use an SQL CASE statement to combine the results of such subexpressions, based on the values of the parameter(s).

We arrive at a final PAR evaluation view that consists of selection conditions and expressions over the parameters (plus, in our specific case, an inexpensive computation of the ROLLUP values over the already-aggregated line items) and the single materialized view corresponding to the PAR. From this pairing of materialized PAR encoding and parameterized PAR evaluation view, any combination of scenarios can be directly and efficiently evaluated.

EXAMPLE 4.3. *Continuing our running example, the parameterized view would be as follows.*

```
SELECT Plan, Quarter, Mo,
    CASE WHEN ?s1 AND ?p1=Quarter THEN Amt1
          ELSE Amt1 - Amt2 END
FROM Scenario1_PAR_Encoding
GROUP BY ROLLUP (Plan, Quarter, C.Mo)
```

Our examples have all been based on SQL, under the assumption that the client is a DBMS. Very similar techniques can be used to map to a spreadsheet such as Microsoft Excel: here Caravan computes the PAR-encoded materialized view in an almost identical fashion, using the conditional primitives (IF/ELSE), aggregate functions, and expressions supported by the spreadsheet. Such a view can then be loaded directly into the spreadsheet, in a way that supports fully autonomous evaluation because all data is pre-encoded. Observe that this is quite different from most existing schemes for supporting spreadsheet views over databases, such as Oracle GoldenGate, in that what-if evaluation leaves the database unaffected. It also differs from techniques that simply copy the contents of a view into a spreadsheet, as what-if scenario changes can be made; and from techniques that copy the source data into the spreadsheet, as we do not need to carry along all of the database data.

5. PROOF-OF-CONCEPT EXPERIMENTS

To demonstrate the potential of PARs for speeding up what-if analysis, we show preliminary numbers for a workload based on our example scenario. We took the original view and scenarios used in Example 1.1. We built a simple data generator largely modeled after the one for TPC-H. The data generator creates 400K customers by randomly concatenating strings for names. Customers are assigned one of two plans and a state uniformly at random. Each plan alternates between two prices in consecutive months. Finally, users are allocated different call durations each month. Our experiments compare the performance of computation and evaluation of PARs versus running the user's view (modified for each scenario) in its entirety. We also considered running times for each scenario using incremental view maintenance. We conducted experiments in two settings. To illustrate server performance we used a 2.83GHz dual Xeon E5440 (i.e., 8-core) Dell PowerEdge 1950 with 8GB RAM, Windows 2008 Server (64b), IBM DB2 UDB 9.5.0.808, and Java JDK 1.6.0_11. We also show timings for the same experiments (other than PAR creation, which is intended to be server-side, and non-incremental view update, which is an order of magnitude slower than incremental update) in a laptop setting, for which we used a Core 2 Duo L9600 machine (dual-core) with 4GB RAM and a 200GB SSD, using Java JDK 1.6.0_29 over MySQL 5.1 on Windows 7 SP1. Experiments are averaged across 9 runs, using prepared statements and a warm cache.

From the results in Table 1, we observe that fully recomputing the view instance based on scenarios (S1–S3) is at least as expensive as the original query (Scenario S1 introduces a union, so it is more costly). Creating the PAR (build PAR) has essentially the same cost as computing the original view instance (the 0.2 sec difference is well within the variance between runs). Incremental maintenance using the base data (incremental-S1 – incremental-S3), where we have local access to the entire database, takes several seconds on the server, and more than 8 seconds on the laptop. This is significantly slower than users expect for an interactive application, and moreover seems implausible in mobile settings. On the other hand, computing scenarios via the PAR (PAR-original-view and PAR-S1–PAR-S3) provides results in under 2.5 milliseconds — a 1000x performance improvement on the server.

We also copied the PAR to Microsoft Excel 2007, using Excel's conditional formulas to compute the result from the PAR, and converting the ROLLUP columns to cumulative totals. The effects of updating the results here were seemingly instantaneous (but difficult to measure due to the interactive user interface).

Our results are still preliminary, but we believe these suggest the speedups are significant enough to make previously-expensive queries into ones that can be computed at interactive speeds. While in general we expect that the user may not be exploring a huge number of potential combinations of scenarios, it is entirely likely that they may be adjusting certain *parameters* many times to "fine tune" their results.

Computation	Server	Laptop
build PAR	11.2 sec	N/A
original view	11.4 sec	4.65 min
PAR-original-view	0.66 msec	2.76 msec
S1	18.3 sec	> 4 min
incremental-S1	7.64 sec	21.4 sec
PAR-S1	2.33 msec	3.33 msec
S2	11.1 sec	> 4 min
incremental-S2	4.17 sec	5.80 sec
PAR-S2	0.64 msec	2.43 msec
S3	10.3 sec	> 4 min
incremental-S3	3.57 sec	8.21 sec
PAR-S3	0.76 msec	1.50 msec

Table 1: Experimental results show significant speedup factors for PARs.

6. RELATED WORK

We discuss related work in two areas: (1) other approaches to answering queries under hypothetical updates and (2) data annotation, which serves as background for our approach.

We have already discussed in the introduction that with unrestricted direct access to the source data or by building a replica of it (e.g., in a warehouse) we can perform what-if analysis with conventional means: running multiple variants of query and performing as needed transactional updates (and then rollbacks). Updates/rollbacks are expensive and cumbersome. Avoiding transactional updates is the goal of the elegant work in Heraclitus/HQL [8, 12] and SESAME [4] which investigates languages that combine querying with hypothetical updates, and their implementation over data warehouses. The salient ingredient in these papers is *rewriting* of queries under hypothetical update scenarios. We use rewriting also but there is a big difference. Suppose a user has n scenarios and thus may be interested in up to 2^n combinations thereof. Heraclitus/HQL/SESAME rewrite the query together with each combination of scenarios separately and run the resulting query. Thus to get 2^n different answers they run 2^n different queries, each of complexity comparable to the original query. Caravan rewrites the query together with original n scenarios producing *one* (provisioning) query. Running this query builds a PAR. Out of this PAR we can obtain, at interactive speeds, any of the desired 2^n answers. As we show in our experiments, for Caravan computing an answer out of a PAR is several orders of magnitude faster than running the original query or its scenario-modified versions, even with incremental maintenance.

Evaluating Datalog queries under hypothetical deletions and insertions was studied in [5]. [1] The scenarios are not separated from the program and, as in Heraclitus/HQL/SESAME, each combination of scenarios leads to a running a separate program. Another

[1]The paper is a also a link to the rich AI literature on "hypothetical reasoning".

related project [15] adds *perspectives* to Essbase/MDX [2, 1] thus enhancing the capabilities of its warehouse server to allow for refinement of OLAP query dimensions.

The concept of *annotating* data with Boolean conditions which is central to our proposed techniques has its roots in the seminal work on *conditional tables* for incomplete databases [13]. We further build upon recent advances in *data provenance* research that uses the "semiring framework" [10]. Specifically, we use the concept of *annotating queries*, introduced in [7]. We also note that *aggregations* are central to what-if analysis and therefore in our example; the mechanism of propagating annotations through aggregation that we used in the generation of PARs was developed in [3].

7. CONCLUSIONS AND CHALLENGES

We have presented an approach for a fast, lightweight, interactive client-based exploration of what-if scenarios, that is centered around the notion of provisioning. We have shown how to compute a Provisioned Autonomous Representation (PAR) that allows the user to explore a set of different what-if scenarios, without resorting to the original database, and without expensive evaluation. Such PARs require little computing power to produce view instances, and are thus well-suited to client-based evaluation in a spreadsheet, on a tablet, or even on a phone. Our initial experimental results show that the proposed method outperforms conventional schemes by orders of magnitude.

There are many essential aspects to consider in future work and we list the most important of these.

The PAR selection problem. One of the major challenges that PARs introduce — as with materialized views and the associated view selection problem — is the question of what autonomous representation(s) to compute for a set of user views, and when they are advantageous. This requires a more comprehensive theoretical understanding of the sizes of PARs (for most views the size of the PAR depends on the number of different scenarios rather than on the database size, but there are formulations where the domain of the parameters may also affect the size). Effective cost estimation techniques are also needed for computation and evaluation. Moreover, our early experience suggests that there are often choices between creating multiple PARs or creating larger PARs, and these have different performance implications. Another important question is how best to materialize PARs for use in answering *multiple* views, as in [17]. Finally, for situations in which the source data itself is distributed, as in [14], there are many questions about where to compute and place PARs. Given that computation becomes more expensive, PARs are actually likely to be even more desirable.

PAR encoding. Our current scheme is fairly primitive for encoding the PAR and the query or spreadsheet that evaluates scenarios over it. We plan to explore different schemes for making the PAR more compact by reusing columns when certain scenarios are mutually exclusive. Additionally, for some scenarios, especially those involving aggregate functions applied over values conditioned on scenarios, it may be more efficient to add further rows to the database, which might reduce the number of additional views to be created. Of course, such decisions may "blow up" the number of tuples and must be done in a cost-based fashion.

Insertions and random choices. Our examples did not include hypothetical *insertions*. One may consider two flavors of insertions in this context. The first, which is easy to incorporate in Caravan, comprises only fully specified insertions of the kind considered in Heraclitus/HQL/SESAME. Such insertions require the analyst to incoporate the new tuples in full detail within the what-if scenarios. Clearly this approach is restricted to a few insertions. For example, within our running PhoneCo example, a possible scenario would be to add a new plan and switch all Pennsylvania customers to it. This involves hypothetically inserting a few tuples into the `Plan` table and hypothetically modifying `Subscr`. Let us also point out that our framework also supports the case when inserted data is from a view of the *existing* data.

However, there are other interesting kinds of hypothetical insertion scenarios. In particular, analysts will want to postulate the addition of large amounts of new data, too large for them to specify in complete detail (when pressed they might say that some of the new data consists of *random* values). Consider the scenario "(S5) What if we acquire 10% more customers in Connecticut?" It sounds reasonable, but what are the plans and the calls of these new customers? The user will not specify these for tens of thousands of records. Instead she might say "distributed randomly like the others" or "plan `A` or `B` with equal probability", etc.

The issue of random choices is not restricted to insertions. Consider another natural question: "(S6) What if 30% of plan `A` customers in Pennsylvania switch to plan `B`?" This is also underspecified: *which* of the customers should we switch? (this matters since the call durations are different). It turns out that Caravan can handle S6 and also random deletions by building a PAR based on *probabilistic conditional tables* [11] which can also be extended to expected values of aggregations.

Scenarios like S5, however, are an interesting challenge for future work. Another interesting challenge for future work is to consider random choices governed by *continuous* distributions. Some such can already be handled with formulas in the provisioning computation while others require a significant extension of the framework.

Specification Languages and Tools. While we have developed an early set of SQL extensions in Caravan to compute a PAR, an additional important facet to consider is the specification of the what-if scenarios by the administrator or the user. We plan to design the formal syntax and semantics of such specification language, building upon the foundations of *annotating queries* [7].

Clearly, specifying what-if scenarios is not an easy task for an analyst both because it requires understanding *what can be changed* and because of the dangers of scenarios that may be inconsistent with each other, may yield different results depending on the order they are applied, or may lead to unwieldy PARs. We discussed already the last issue above. More generally, to help the analyst we will further pursue the implementation of interactive *tools* that aid users in specifying the scenarios, point out potential inconsistencies provide feedback on the impact of each new scenario on the size or cost of the PAR.

Acknowledgments

We thank the reviewers for their helpful comments. This research was funded in part by NSF grants IIS-1217798 and CNS-1065130, by the Binational (US-Israel) Science Foundation, by the Israeli Ministry of Science, and by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007-2013) / ERC grant MoDaS, agreement 291071. Val Tannen is grateful to EPFL for the excellent sabbatical support received.

8. REFERENCES

[1] Multidimensional expressions (mdx) reference. Available at `http://msdn.microsoft.com/en-us/library/ms145506.aspx`.

[2] Oracle Essbase overview. Available at http://www.oracle.com/technetwork/middleware/essbase/overview/index.html.

[3] Yael Amsterdamer, Daniel Deutch, and Val Tannen. Provenance for aggregate queries. In *PODS*, 2011.

[4] Andrey Balmin, Thanos Papadimitriou, and Yannis Papakonstantinou. Hypothetical queries in an OLAP environment. In *VLDB*, 2000.

[5] Anthony J. Bonner. Hypothetical datalog: Complexity and expressibility. *Theor. Comput. Sci.*, 76(1), 1990.

[6] Ronald Fagin, Phokion Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. *TCS*, 336:89–124, 2005.

[7] J. Nathan Foster, Todd J. Green, and Val Tannen. Annotated XML: queries and provenance. In *PODS*, 2008.

[8] Shahram Ghandeharizadeh, Richard Hull, and Dean Jacobs. Heraclitus: Elevating deltas to be first-class citizens in a database programming language. *TODS*, 21(3), 1996.

[9] Todd J. Green and Zachary G. Ives. Recomputing materialized instances after changes to mappings and data. In *ICDE*, 2012.

[10] Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, 2007.

[11] Todd J. Green and Val Tannen. Models for incomplete and probabilistic information. In *EDBT Workshops*, 2006.

[12] Timothy Griffin and Richard Hull. A framework for implementing hypothetical queries. In *SIGMOD*, 1997.

[13] Tomasz Imielinski and Witold Lipski. Incomplete information in relational databases. *JACM*, 31(4), 1984.

[14] Zachary G. Ives, Todd J. Green, Grigoris Karvounarakis, Nicholas E. Taylor, Val Tannen, Partha Pratim Talukdar, Marie Jacob, and Fernando Pereira. The ORCHESTRA collaborative data sharing system. *SIGMOD Rec.*, 2008.

[15] Laks V. S. Lakshmanan, Alex Russakovsky, and Vaishnavi Sashikanth. What-if OLAP queries with changing dimensions. In *ICDE*, 2008.

[16] Bin Liu and H. V. Jagadish. A spreadsheet algebra for a direct data manipulation query interface. In *ICDE*, 2009.

[17] Hoshi Mistry, Prasan Roy, S. Sudarshan, and Krithi Ramamritham. Materialized view selection and maintenance using multi-query optimization. In *SIGMOD*, 2001.

[18] Andrew Witkowski, Srikanth Bellamkonda, Tolga Bozkaya, Gregory Dorman, Nathan Folkert, Abhinav Gupta, Lei Sheng, and Sankar Subramanian. Spreadsheets in RDBMS for OLAP. In *SIGMOD*, 2003.