Dear PVLDB Chair and Referees:

We thank the reviewers for the very helpful feedback on our paper. We have tried to address all of the listed concerns and have included references to the revised text in the cover letter. To summarize the major revisions:

1. We revised our background section (Section 2.1) to include a detailed running example which is referenced in Examples 1-6 after each major concept.

2. We revised our overview section (Section 3) to formalize our problem setting, assumptions, terminology, and key prerequisite concepts in our work and included more formal problem statements.

3. The sampling section (Section 4) included a detailed discussion about the challenges and new ideas (Section 4.1) and clarified a significant reviewer request about the definition of "primary keys".

4. The result estimation (Section 5) has been revised to include itemized descriptions of all of the algorithms and is self-contained with respect to our prior work.

5. Experiments (Section 7) have been revised to merge redundant experimental results. Rather than presenting two different experimental results, we use our real dataset to evaluate SVC on group by aggregate views.

We first will detail our changes in response to the meta reviewer comments and then address the detailed reviewer comments subsequently.

[Meta Reviews]

M1. The definitions and discussions, which are currently presented in a very hand-waiving manner, need to be replaced with their formal counterparts. The presentation should be revised, also to avoid the continuous references to the tech report for details. Please see the detailed comments E1 of reviewer 1, and C1,C2, C4, C5, and C6 of reviewer 2 for more details.

Responses: We have significantly clarified the presentation of the concepts and the algorithms. Section 3.1 (Notations and Definitions) has been expanded to formally present the core preliminaries of our work: Materialized Views, Staleness, Relational Expressions for Maintenance, and Uniform Sampling. Section 3.2 presents an itemized workflow of SVC and formal descriptions of the problems that SVC solves. In our sampling section (Section 4), we replace the informal descriptions with their formal counterparts including Definition 1 (Provenance), Proposition 1 (Primary Key based Provenance), and Property 1 (Correspondence). We summarize the intutitions in these formal concepts with Example 3 and Example 4. We revised Section 5 to have an itemized description of the estimation algorithms proposed in this work. Additionally, the technical report is now only cited in the Experiments section with reference to details in the experimental setup and materialized view choice.

M2. There are several assumptions and restrictions that are not spelled out clearly in the first part of the paper. It should be clarified how much they limit the applicability of the proposal. The real-world scenarios used is very interesting, but do the techniques apply in other popular applications, where the assumption in sec 4.2 may not hold?

Responses: We have revised the presentation of the work to be more explicit about limitations. We added the following paragraph before the problem statements in Section 3.1:

> In SVC, we explore the problem of approximate aggregate query processing on stale materialized views using a data-cleaning approach. We assume that these materialized views are periodically maintained and thus are stale in between maintenance periods. The focus of this paper is analytic workloads where the typical query on the view is a group by aggregate. SVC provides a framework for increased query accuracy for a flexible maintenance cost that can scale with system constraints.

We believe this concisely summarizes our problem domain and applicability of our proposal. Furthermore, we have clarified that the primary key definition proposed in 4.2 (Section 4.3 in the revised paper) is not an assumption but a generation procedure. For the relational expressions described in the paper (select, project, join, aggregate, union, difference), if there is a unique primary key for the base relations, we can ensure that any derived relation also has a unique primary key by the rules described in Definition 2. If the base relations do not have primary keys, then we can add an extra column to the relation that assigns each row a unique id. We added Example 2 to describe this process concretely.

M3. There are recent proposals in data cleaning over materialized views that tackle an orthogonal problem: While the setting is different, work has been done on how to use a different statistical measure (sensitivity analysis) to tackle similar technical problems (sec 5.1.1, sec 6.3). The authors can find related techniques in the recent work on data cleaning over views. It would be useful to have a technical discussion of how the proposed techniques can be applied in this related setting and vice versa (an experimental evaluation is not needed): (1) Wu and Madden. Scorpion: Explaining Away Outliers in Aggregate Queries. PVLDB 2013, (2) Chalamalla et al. Descriptive and prescriptive data cleaning. SIGMOD 2014, (3) Meliou et al. Tracing data errors with view-conditioned causality. SIGMOD 2011

Responses: We have added a paragraph to our related work (Section 8) contrasting these works from ours. While, all three of these works address the issue of dirty materialized views and using lineage to trace the errors to base data, they are not directly applicable in the materialized view setting. These three works require an explicit specification of erroneous rows in a materialized view. Identifying whether a row is erroneous requires materialization and thus specifying the errors is equivalent to full incremental maintenance. Instead, in our work, we use the formalism of a "maintenance strategy", the relational expression that updates the view, and uses this expression to understand how errors occur and how to sample the view while preserving uniformity. Additionally, for this reason, in our outlier indexing method, we propagate outliers up from base data rather than tracing errors back to base data. From the text:

> For example, Wu and Madden [31] studied explanations for outliers in group-by aggregate queries. The technique used to trace the provenance of an outlier row in a result is very similar to the technique proposed in this work. However, this work addressed a different cleaning problem from SVC. In SVC, we look to find a relational expression that update an erroneous row such that it is correct. In [31], the authors remove base data such that the row is no longer an outlier. Also, Challamalla et al. [4] proposed a technique for specifying errors as constraints on a materialized view and proposing changes to the base data such that these constraints can be satisfied. In our setting, we do not know these constraints a priori and we would have to materialize the entire up-to-date view to be able to specify

> which rows are erroneous. Finally, the work by Meliou et al. [21] also explores tracing lineage through materialized views. Like [31], it requires an explicit specification of the errors and also does not suggest a cleaning operation to fix these errors.

M4. The experiments sections needs to be improved to include comparison with relevant work, more details and explanation. Please look at the detailed comments E2, E4, and E5 of reviewer 1, and B1 and B2 of reviewer 2 for more details.

Responses: We have addressed all of the details from reviewer comments in the reviewer section of the cover letter. To summarize, we cited the algorithm that we used for Incremental View Maintenance which is the change-table (called a delta table in our work) algorithm described in Incremental maintenance of aggregate and outer join expressions by Gupta and Mumick 2006. We further clarified our contribution for the two compared query processing approaches, SVC+CORR and SVC+AQP. Both techniques use our Stale View Cleaning technique but have a different result estimation procedure. There were also reviewer concerns about selectivity (for which we added a theoretical analysis) and update size (for which we clarified the experiment in which those results can be seen). In addition, two reviewers suggested removing our detailed evaluation on a distributed platform to save space. We have consolidated our experiments such that we still present accuracy and performance results for the real dataset from Conviva Inc. but do not describe the details of our implementation and deployment on Apache Spark.

M5. The paper's main motivation is that eager IVM cannot keep up with the rate of incoming updates. However, there have been approaches in the literature (most notable DBToaster) suggesting that this issue can be resolved by accelerating IVM. Do you think that there is still a need of SVC? What is the rate of updates over which a system such as DBToaster cannot keep up with the updates?

Responses: In the paper (Section 2.1), we have added the following clarification and motivation of the work:

> There has been significant research on fast MV maintenance algorithms, most recently DBToaster [18]. However, even for these optimized systems, some materialized views are computationally difficult to maintain. In their evaluation of DBToaster, Koch et al. [18] reported a three order of magnitude variation in maintenance throughput over the 22 TPCH queries defined as MVs. Furthermore, in real deployments, it is common to use the same infrastructure to maintain multiple materialized views (along with other analytics tasks) adding further contention to computational resources and reducing overall maintenance throughput. SVC is complementary to existing maintenance algorithms, whether 1st order like classical change-table IVM or higher order like DBToaster, provided they are specified in standard relational algebra. Through the use of sampling, we provide the user another tool to flexibly manage and schedule maintenance of MVs for aggregate analytics.

M6. Typos and other minor issues as listed by E6 of reviewer 1 and C7, C8, C9, C10 and C11 of reviewer 2 should be addressed.

Responses: We thank the reviewers for their careful read of the paper, and have addressed these issues.

[Reviewer 1]

B1. There are several assumptions and restrictions that are not spelled out clearly in the first part of the paper. It should be clarified how much they limit the applicability of the proposal. The real-world scenarios used is very interesting, but do the techniques apply in other popular applications, where the assumption in sec 4.2 may not hold?

Responses: This review is addressed above in Meta Review Section (M2).

B2. It is great to see many technical and engineering contributions, but the paper is very dense and hard to read. The first clear example of what is going on appears at page 4, after the reader had tried hard to understand the problem statement in sec 3.3.1. The presentation should be revised, also to avoid the continuos references to the tech report for details.

Responses: We have revised the presentation of the work to be easier to follow. In Section 2.1, we describe a running example of Video Streaming Log Analysis. In Section 3, there are two detailed examples of the concepts presented. Example 1 describes the terminology and prerequisite concepts in terms of a concrete use case, and Example 2 illustrates the end-to-end workflow of SVC. In Section 4, we add Example 3 to clarify the reviewer concern about the applicability of primary key lineage in this setting and Example 4 to show how we can optimize sampling of a materialized view in a real application. In Section 5, we add Example 5 to describe our query processing approaches. In Section 6, we add Example 6 to describe how outlier indexing would be used in practice.

We have further minimized the references to the technical report. The technical report is now only for details in the experimental setup. The theoretical presentation of this work is now self contained with respect to our prior work.

B3. There are recent proposals in data cleaning over materialized views that tackle an orthogonal problem: given a view, they clean a sample of its data and go back to the base relations to identify useful explanations. While the setting is different, work has been done on how to use a different statistical measure (sensitivity analysis) to tackle similar technical problems (sec 5.1.1, sec 6.3). Given the data cleaning angle of the proposal, a comparison with these techniques is relevant in this work.

Responses: We address this issue in the Meta Review Section (M3).

E1. I would recommend the authors to revise the presentation of the paper to make it more accessible to the readers, for example with more examples and by limiting the tech report for relevant information. While it is great to show great engineering effort, I think it would be beneficial for this work to deliver more clearly what are the key intuitions and novel ideas. For example, I am not sure I understand the need to conduct the experiment on a distributed platform, as it doesn't touch any of the key contributions of the work. Of course, it is hard to add examples, comparisons, and clarifications without removing something, but in this case I'd say this experiment could be moved to the tech report to leave more room to clarify the basic ideas of the paper and make it more self-contained (e.g., the discussion on CLT with the ref to [36]).

Responses: We now summarize our contributions in the introduction as follows:

> (1) we formalize maintenance of a sample MV as a data cleaning operation on the sample, (2) we propose an optimization technique that materializes the clean sample efficiently while preserving correctness, (3) we derive a query processing approach to answer aggregate queries accurately using the clean sample, (4) we propose an outlier index to reduce sensitivity to

> skewed datasets, and (5) we evaluate our approach on real and synthetic datasets confirming that indeed sampling can reduce view maintenance time while providing accurate query results.

To make the presentation more accessible, we have revised the work with the clarifying examples described above (B2), and introduced a itemized summary of all of the components in Section 3.2. We have also revised Section 4 (Stale View Cleaning) of the paper with the following clarifications: introducing problem specific challenges (Section 4.1) and a clarified presentation of Provenance (Section 4.2-3). As the reviewer suggested, we have limited the use of the technical report to experimental engineering details. In our submission, many of the details in Section 5 (Query Result Estimation) were omitted and discussed in the technical report. In addition to the clarifying examples described for review B2, we have made the following revisions to Section 5 to make it more accessible: (1) we present itemized algorithms for our query result estimation approaches, (2) we provide SQL descriptions of confidence interval calculations via the CLT, and (3) we present a simpler taxonomy of different families of aggregate queries and their properties.

E2. I would anticipate a discussion on the context in which the approach works. I would also like to understand why it is hard to go beyond the assumptions with a more general discussion. Right now there are limitations spread over the paper:

Responses: We address the first part of this question in the Meta Review section (M2). For the second part of the question, we expanded the Limitations section (Section 9) at the end of the paper:

> While our experiments show that SVC works for a variety applications, there are a few limitations which we summarize in this section. There are three primary limitations for SVC: class of queries, types of materialized views, and the specification of maintenance strategy. In this work, we primarily focused on aggregate queries and showed that accuracy decreases as the selectivity of the query increases. Sampled-based methods are fundamentally limited in the way they can support "point lookup" queries that select a single row. This is predicted by our theoretical result that accuracy decreases with $\frac{1}{p}$ where p is the fraction of rows that satisfy the predicate. In terms of more view definitions, SVC does not support views with ordering or "top-k" clauses, as our sampling assumes no ordering on the rows of the MV and it is not clear how sampling commutes with general ordering operations. SVC also requires the maintenance strategy to be parametrized in terms of relational algebra which may not always be possible. Tools like DBToaster achieve some of their performance gains by code generation and not specifying maintenance in SQL.

E2-1. In 3.3.2 for the sql I was also expecting an experiment to see different quality results depending on the selectivity of the query

Responses: We included a theoretical analysis of selectivity in Section 5.3.3:

> Let p be the selectivity of the query and k be the sample size; that is, a fraction p records from the relation satisfy the predicate. For these queries, we can model selectivity as a reduction of effective sample size $k \cdot p$ making the estimate variance: $O(\frac{1}{k*p})$. Thus, the confidence interval's size is scaled up by $\frac{1}{\sqrt{p}}$. Just like there is a tradeoff between accuracy and maintenance cost, for a fixed accuracy, there is also a tradeoff between answering more selective queries and maintenance cost.

We believe these results are predictable as for a fixed p, $\frac{1}{\sqrt{p}}$ is just a constant scaling on the accuracy results. In our experiments, we randomly generated queries with a variety of different selectivities described in Section 7.1.1:

> For each of the views, we generated *queries on the views*. Since the outer queries of our views were group by aggregates, we picked a random attribute a from the group by clause and a random attribute b from aggregation. We use a to generate a predicate. For each attribute a, the domain is specified in the TPCD standard. We selected a random subset of this domain, e.g., if the attribute is country then the predicate can be countryCode > 50 and countryCode < 100. We generated 100 random `sum`, `avg`, and `count` queries for each view.

For example, in Figure 4, the average selectivity was 24.1%. If we chose twice as selective queries, the errors would scale by up $\sqrt{2} \approx 1.4$.

E2-2. In 4.2 for the PK requirement this seems very strong and not realistic in many applications: what if this assumption does not hold? would AQP also fails?

Responses: We have clarified that the primary key definition proposed in 4.2 (Section 4.3 in the revised paper) is not an assumption but a generation procedure. For the relational expressions described in the paper (select, project, join, aggregate, union, difference), if there is a unique primary key for the base relations, we can ensure that any derived relation also has a unique primary key by the rules described in Definition 2. If the base relations do not have primary keys, then we can add an extra column to the relation that assigns each row a unique id. We added Example 2 to describe this process concretely.

E2-3. In 6.1 for the background knowledge is it realistic to have the user defining all these indexes? can also the traditional incremental solution benefit for a similar optimization? you should clarify if the experiments before 7.2.4 are done with or without the indexing. If they all done without the indexing, it seems that your methods does not really needed this optimization

Responses: We have added clarification on how these indices may be constructed in Section 6.1:

> There are many approaches to select a threshold. We can use prior information from the base table, a calculation which can be done in the background during the periodic maintenance cycles. If our size limit is k, for a given attribute we can select the top-k records with that attributes. Then, we can use that top-k list to set a threshold for our index. Then, the attribute value of the lowest record becomes the threshold t. Alternatively, we can calculate the variance of the attribute and set the threshold to represent c standard deviations above the mean.

We used this approach in our experiments (Section 7.2.4) and list the tradeoff between outlier index size and improvements in query result accuracy. This outlier optimization is only relevant to sampling based approaches as those can be sensitive to the presence of outliers. Traditional IVM cannot benefit from this approach. We have also clarified that none of our experiments before 7.2.4 used an outlier index. The caveat is that these experiments were done with moderately skewed data with Zipfian parameter = 2, if this parameter is set to 4 then the 75% quartile query estimation error is nearly 20% (Figure 11). Outlier indexing always improves query results as we are reducing the variance of the estimation set, however, this reduction in variance is largest when there is a longer tail. In this setting, outlier indexing significantly helps for both SVC+AQP and SVC+CORR.

E3. As mentioned in B3, the authors can find related techniques in the recent work on data cleaning over views. It would be useful to have a technical discussion of how the proposed techniques can be applied in this related setting and vice versa (an experimental evaluation is not needed): - Wu and Madden. Scorpion: Explaining Away Outliers in Aggregate Queries. PVLDB 2013 - Chalamalla et al. Descriptive and prescriptive data cleaning. SIGMOD 2014 - Meliou et al. Tracing data errors with view-conditioned causality. SIGMOD 2011

Responses: This discussion is clarified in the Meta Review section (M3) and we have added a discussion to our related work.

E4. I am not sure I got why you are not reporting the execution times for AQP in fig 7.a, 9.a, 11.a. It would be interesting to have it to understand better the trade-off.

Responses: To address this comment, we clarified the contributions of our approach. In our Stale Sample View Cleaning problem, we study how to efficiently maintain a sample of a materialized view. After maintenance, there are two query result estimation approaches that can be used: SVC+CORR and SVC+AQP. Thus, the maintenance time for both SVC+CORR and SVC+AQP is the same as they both use SVC as an underlying sample maintenance framework. In fig 7.a, 9.a, and 11.a, we measure the maintenance time so there is no need to compare the methods. We clarify this point in the Section 7.1.2 of the experiments: We use the following notation to represent the different approaches:

> **SVC+AQP:** We maintain a sample of the materialized view using SVC and estimate the result with SVC+AQP.
>
> **SVC+CORR:** We maintain a sample of the materialized view using SVC and process queries on the view using SVC+CORR.

E5. I got the justification for Def 1 only after reading the rest of the paper (e.g., sec 4.4). While it is natural, the first time I read it I was wondering why don't model it as a graph homomorphism, or any common, existing definition to describe a transformation between two instances. It would be easier to understand and justify.

Responses: We have clarified this point by re-arranging the text. The correspondence definition is now in Section 4.5, where we carefully explain the intuition behind this property. Correspondence formalizes the link between the unique keys in sample of a stale materialized view and a sample of an up-to-date materialized view. We also clarified the correspondence formally in Property 1 (Section 4.5), where we define four conditions: uniformity, removal of superfluous rows, sampling of missing rows, and key preservation for updated rows.

E6. typos: (1) sec 1: which USES APPLIES data (2) sec 3.2: STRATFIED sampling (3) some sentences need revised punctuation. For example, in sec 6: "The intuition is that there.... outliers" (4) missing s at the end of 7.1.1 (5) sec 7.2.1: , instead of . after view

Responses: We have corrected these typos.

[Reviewer 2]

WP1. Major flaws in the presentation: Most of the concepts and algorithms are introduced using words (and on top of that formulations that can be misinterpreted), making it hard to completely follow and be able to replicate the proposed approach. Other presentation issues include lack of examples, and introduction of the approach not in a standalone way but through comparison to previous work by the authors.

Responses: We have added the following formalization to clarify the concepts presented in the paper. In Section 3.1, we formalize the prerequisite concepts in this work: materialized view maintenance, staleness data error, unique primary keys, and uniform sampling. We conclude Section 3.1 with a detailed discussion of our running example making the formalization concrete. In Section 3.2, we present an itemized formal workflow of the entire SVC system. This introduces the two problems addressed in this work: Stale Sample View Cleaning and Query Result Estimation. In Section 4, we add Definition 1-3, Proposition 1, and Property 1 to formally present the key concepts in our work. In addition, there are two examples in this section to clarify the concepts. In Section 5, we present itemized descriptions of the algorithms for query result estimation and present the confidence interval calculation in terms of SQL expressions. We have minimized references to our prior work, SampleClean. We introduce this once and describe the key contributions that build on the SampleClean theoretical framework.

WP2. Support for non-aggregate queries seems like an afterthought: It is only briefly discussed in two paragraphs in Section 5.3 and it is not clear how it would work (and how the error in such a case could be measured). As far as I could tell no experiments were executed on such queries.

Responses: Due to space restrictions, we have removed our discussion of support for non-aggregate queries as it is not essential to our work. SVC can support error estimation for SELECT queries (with low selectivity) by estimating how many row are missing due to sampling.

WP3. Motivation is slightly weak, given that recent IVM approaches, such as DBToaster have suggested that IVM can be greatly accelerated, making it thus much easier to keep up with changes to the base tables.

Responses: In Meta Review 5, we clarify that there are some views for which even DBToaster is slow. Sampling, as proposed in this work, reduces the cost of maintenance and is complementary to the choice of maintenance algorithm.

A. The paper's main motivation is that eager IVM cannot keep up with the rate of incoming updates. However, there have been approaches in the literature (most notable DBToaster) suggesting that this issue can be resolved by accelerating IVM. Do you think that there is still a need of SVC? What is the rate of updates over which a system such as DBToaster cannot keep up with the updates?

Responses: We address this issue in the Meta Review Section (M5). We first clarify that SVC is complementary to the choice of maintenance algorithm. Sampling has the potential to reduce maintenance costs for any algorithm (provided it can be specified in relational algebra) by reducing the number tuples processed. In the specific case of DBToaster, over the TPCH queries there was a 3 order of magnitude variation in maintenance throughput. If this data grows, is distributed, or resources are contended by other tasks, this latency can easily grow significantly. While approaches like DBToaster greatly accelerate IVM, there are some views that are slow to maintain just due to processing each tuple for aggregates and joins. Sampling reduces the number of tuples processed and trades off accuracy in these settings where eager maintenance is expensive.

B1. What is the exact IVM algorithm that is used in the experiments?

Responses: The algorithm that we used for Incremental View Maintenance is the change-table (called a delta table in our work) algorithm described in Incremental maintenance of aggregate and outerjoin expressions by Gupta and Mumick 2006. This is cited and clarified in our experiments section. From the text

> The incremental maintenance algorithm used in our experiments is the "change-table" technique proposed in [16]. We implement incremental view maintenance with an "update...on duplicate key insert" command. We implement SVC's sampling operator with a linear hash written in C that is evoked in MySQL as a stored procedure. In all of the applications, the updates are kept in memory in a temporary table, and we discount this loading time from our experiments.

B2. In the join view experiment, you report the accuracy of SVC for 10% sample size. What is the update size in this case? It would be great to see how the update size (which has been shown before to affect the speedup) affects also the accuracy of the algorithm.

Responses: We clarified that the update size was 1GB corresponding to 10% of the base data (Figure 5). Figure 6b illustrates the tradeoff between update size and the accuracy of the algorithms. SVC+CORR grows in error proportional to the update rate, while SVC+AQP stays constant. The break even point is when the update size is about 30% of the base data.

C1. Formulations: Most of the concepts are introduced very informally in words, in a way that makes it hard to fully understand what is meant. The use of terminology is very lax as well.

Responses: See Meta Review Section (M1) for the summary of changes made to the concepts.

C1-1. Here are a few examples: (a) definition 1 is not formal enough. For instance, what does it mean "required a delete"? Although in this case one can understand what is meant, it should be presented in a more rigorous way,

Responses: We revised the definition of staleness data error in Section 3.1. We included both an intuitive definition and a formal definition for this concept:

> **Staleness as Data Error:** The consequences of staleness are incorrect, missing, and superfluous rows. Formally, for a stale view S with primary key u and an up-to-date view S':
>
> - **Incorrect:** Incorrect row errors are the set of rows (identified by the primary key) that are updated in S':
> $\{\forall u \in S : (\exists u \in S' \wedge (\sigma_u(S) \neq \sigma_u(S')))\}$
> - **Missing:** Missing row errors are the set of rows (identified by the primary key) that exist in the up-to-date view but not in the stale view:
> $\{\forall u \in S' : \nexists u \in S\}$
> - **Superfluous:** Superfluous row errors are the set of rows (identified by the primary key) that exist in the stale view but not in the up-to-date view :
> $\{\forall u \in S : \nexists u \in S'\}$

C1-2. The term "query correction" in Section 3.3.2 is misleading since it is not the query statement that is corrected but the query result

Responses: We have also revised the query correction term to "Query Result Estimation" which we feel is more accurate.

C1-3. In the last paragraph in Section 7.1.2, the views are referred to interchangeably as "views" and "dataset". I would suggest to have a more formal introduction of the concepts and establish a terminology that is used consistently throughout the paper.

Responses: We corrected the inconsistencies in term usage, using the term dataset ONLY to refer to the base data of the experimental data from TPCH and Conviva.

C1-4. If you end up needing more space in the process a few places you could compress are the following: (a) the algebra in Section 3.1, since it is standard relational algebra, (b) Section 7.3.2, which although interesting is I believe less important than a formal representation of the core concepts, (c) Section 7.2.3 (together with the corresponding graphs) which could be replaced just by a datapoint showing that if hashing cannot be pushed down, the resulting speedup is limited.

Responses: We have also incorporated the reviewers space saving suggestions by revising the presentation of the relational algebra, experiment 7.3.2, and consolidated our experiment on real data and the TPCD data cubing example.

C2. Algorithms: Please try to introduce the algorithms formally (e.g., through pseudocode). Also consider adding a more formal description of the entire workflow followed by SVC apart from Figure 1 (something close to the itemization in Section 5.2 but with formal notation instead).

Responses: We have included an itemization for all of the algorithms in Section 5, including the SQL for calculating the bounds for `sum`, `avg`, and `count` and the pseudocode for the bootstrap algorithm to bound general aggregate queries. In addition, in Section 3.2 we added an itemized description of the full workflow of SVC.

C3. Related Work: Currently comparisons to related work (especially SampleClean but also AQP and SAQP) are dispersed in various places throughout the paper, breaking its flow. In many cases SVC is not introduced on its own but through comparisons to SampleClean (e.g., in Sections 3.3.2, 5.1, 5.2, etc). I would suggest to introduce instead SVC without reference to SampleClean and if a comparison is needed, to limit it either to Section 2 or to a short discussion at the end of each (sub)section.

Responses: We have addressed the reviewers suggestion and made this comparison more concise and introduced SVC on its own. SampleClean is cited once in Section 2.2, where we introduce SVC and explain the challenges in the materialized view problem setting that differ from the problem studied in SampleClean. In the remaining paper, references to SampleCleans algorithms and approaches have been removed. Section 5 has been greatly revised to present SVC on its own. We present a self contained theoretical discussion of the Central Limit Theorem and how to calculate the confidence intervals. We only cite SampleClean once in Section 5.2 in reference to other approximate query processing techniques that use analytic confidence intervals.

C4. Examples: Please add examples after the introduction of each concept/algorithm to help the reader follow them. For instance, present the correction generated for the running example in Section 5.1. Similarly, show the generated plan in the presence of indexes in Section 6.2.

Responses: We introduced a running example in Section 2.1 based on our experimental dataset. In Section 3.1, we used this running example to clarify our prerequisite concepts and terminology. In Section 3.2, we give an intuitive end-to-end example of the entire workflow. In Section 4.3, we use this example to describe the primary key generation method. In Section 4.4, we describe our hash pushdown optimization. In Section 5.2, we use the example to describe our query result estimation approaches. In Section 6.4, we describe a concrete example of how to use the outlier index to generate a query result estimation plan.

C5. Figures/References: Please increase the size of both figures and references, as they are currently extremely hard to read. In the case of figures you may be able to achieve this simply by using more concise captions.

Responses: With our saved space we have increased the size of images and captions.

C6. Abbreviations: Please make sure that you have introduced all abbreviations before you use them and remind the reader of their meaning if they have been defined in previous sections. For instance, (a) SAQP used in Section 5.2 has not been defined and (b) AQP used in Section 5.1 has just been defined in passing in Section 2.1 and should probably be re-introduced.

Responses: We have taken the reviewers suggestion and clarified these acronyms in Section 2 and Section 5.

C7. p. 1, col. 1, last par.: "making incremental maintenance infeasible" $->$ You probably mean "making eager incremental maintenance infeasible"

Responses: We have made this revision.

C8. The primary key of the result of a union, intersection and difference between R_1 and R_2 is erroneously defined as the primary key of R. It should instead be expressed in terms of R_1 and R_2.

Responses: We have made the following revisions to fix this definition: (1) $R_1 \cup R_2$: Primary key of the result is the union of the primary keys of R_1 and R_2, (2) $R_1 \cap R_2$: Primary key of the result is the intersection of the primary keys of R_1 and R_2, and (3) $R_1 - R_2$: Primary key of the result is the primary key of R_1

C9. Theorem 2: I could not parse the 2nd sentence of the theorem. Please rephrase!

Responses: We added a clarification to Section 5.2.4 to simplify the discussion of optimality. It is now framed as a discussion of conditions under which our technique is optimal rather than an absolute claim of optimality. The relevant text is now phrased as follows:

> A sampled relation R defines a discrete distribution. It is important to note that this distribution is different from the data generating distribution, since even if R has continuous valued attributes R still defines a discrete distribution. Our population is finite and we take a finite sample thus every sample takes on only a discrete set of values. In the general case, this distribution is only described by the set of all of its values (i.e no smaller parametrized representation). In this setting, the sample mean is an MVUE. In other words, if we make no assumptions about the underlying distribution of values in R, SVC+AQP and SVC+CORR are optimal for their respective estimates ($q(S')$ and c).

C10. p. 8, col. 2, par. 4: I could not parse the sentence "We remove views... or are static". Please rephrase!

Responses: We clarified this statement in the following way: 10 out of the 22 sets of views can benefit from SVC. For the 12 excluded views, 3 were static (i.e, this means that there are no updates to the view based on the TPCD workload), and the remaining 9 views have a small cardinality not making them suitable for sampling.

C11. Typos/Minor syntactic errors: (1) p. 2, col. 2, par. 5: "insertions into Log which are cached" (2) p. 3, col. 1, example: "then the following expressions are needed" (3) p. 3, col. 1, last line: "Stratified sampling" (4) p. 3, col. 2, first par. of Section 3.3.2: "Given a query q which has been applied to the stale view q(S) giving a stale result, out query" (5) p. 5, col. 1, first par.: "there is an equality outer join" (6) p. 5, col. 1, par. 2: "Foreign Key Join" (7) p. 5, col. 2, last par.: "A case statement is defined as follows: We define pred(*)" (8) p. 7, col. 1, first par. of Section 6: "when the sample contains an outlier" (9) p. 7, col. 2, par. 2: "we can find the records with the top k attribute values" (10) p. 8, col. 2, par. 4: "and use those as our materialized views" (11) p. 8, col. 2, par. 5: Change the symbols in the two predicates involving countryCode (12) p. 8, col. 2, par. 2: "For small update sizes, the speedup is smaller, 6.5x for a 2.5% (250GB) update size": it should probably be "MB" instead of "GB" (13) p. 12, col. 1, par. 4: "if that is a black box"

Responses: We have made these changes to the text.

[Reviewer 3]

Theorem 2 is based on a very naive assumption. The assumption that nothing else is known about the distribution is false in this setting. Given the data in the materialized view, a pretty good a priori estimation of the distribution of the data can be made. Given this estimation, the estimator (MVUE) that best fits the distribution should be chosen. It is not enough to just separate out some outliers.

Responses: We thank the reviewer for this detailed comment and clarified the concepts presented in Section 5.2.4 as more a discussion about optimality (the conditions under which the presented approach is optimal) rather than an absolute claim of optimality. We further revised that our query result estimation algorithms (SVC+AQP and SVC+CORR) are complementary to the choice of estimator and if the data distribution warrants a different estimator with lower variance SVC+CORR and SVC+AQP inherit that optimality property. From the text:

> A sampled relation R defines a discrete distribution. It is important to note that this distribution is different from the data generating distribution, since even if R has continuous valued attributes R still defines a discrete distribution. Our population is finite and we take a finite sample thus every sample takes on only a discrete set of values. In the general case, this distribution is only described by the set of all of its values (i.e no smaller parametrized representation). In this setting, the sample mean is an MVUE. In other words, if we make no assumptions about the underlying distribution of values in R, SVC+AQP and SVC+CORR are optimal for their respective estimates ($q(S')$ and c). Since they estimate different variables, even with optimality SVC+CORR might be more accurate than SVC+AQP and vice versa.
>
> However, if we do know more about the distribution, this optimality is not true in general. The intuitive problem is that if there are a small number of parameters that completely describe the discrete distribution there might be a way to reconstruct the distribution from those parameters rather than estimating the mean.

As a simple counter example, if we knew our attributes were exactly on a line, a sample size of two is sufficient to answer any aggregate query. However, even for many parametric distributions, the sample mean estimators are still MVUEs, e.g., poisson, bernouilli, binomial, normal, exponential. It is often difficult and unknown in many cases to derive an MVUE other than a sample mean. Furthermore, the sample mean is unbiased for all distribution, but it is often the case that alternative MVUEs are biased when the data is not exactly from correct model family (such as our example of the line). Our approach is valid for any choice of estimator if one exists, even though we do the analysis for sample mean estimators and this is the setting in which that estimator is optimal.

Stale View Cleaning: Getting Fresh Answers from Stale Materialized Views

Sanjay Krishnan, Jiannan Wang, Michael J. Franklin, Ken Goldberg, Tim Kraska [†]
UC Berkeley, [†]Brown University
{sanjaykrishnan, jnwang, franklin, goldberg}@berkeley.edu
tim_kraska@brown.edu

ABSTRACT

Materialized views (MVs), stored pre-computed results, are widely used to facilitate fast queries on large datasets. When new records arrive at a high rate, it is infeasible to continuously update (maintain) MVs and a common solution is to defer maintenance by batching updates together. Between batches the MV becomes increasingly stale with incorrect, missing, and superfluous rows leading to increasingly inaccurate query results. We propose Stale View Cleaning (SVC) which addresses this problem from a data cleaning perspective. In SVC, we efficiently clean a sample of rows from a stale MV, and use the clean sample to estimate aggregate query results. While approximate, the corrected query results reflect the most recent data. As sampling can be sensitive to long-tailed distributions, we further explore an outlier indexing technique to give increased accuracy when the data distributions are skewed. SVC complements existing deferred maintenance approaches by giving accurate and bounded query answers between maintenance. We evaluate our method on a real dataset of workloads from the TPC-D benchmark and a real video distribution application. Our experiments confirm our theoretical results: (1) cleaning an MV sample is more efficient than full view maintenance, (2) the corrected results are more accurate than using the stale MV, and (3) SVC can be efficiently integrated with deferred maintenance.

1. INTRODUCTION

Materialized views (MVs), stored pre-computed results, are a well-studied approach to speed up queries on large datasets [7,15, 20]. During the last 30 years, the research community has thoroughly studied MVs for traditional query processing and recently for more advanced analytics based on linear algebra and machine learning [23,33].

However, when the underlying data is changed MVs can become *stale*; the pre-computed results do not reflect the recent changes to the data. One solution would be to recompute the MV each time a change occurs; however, in many cases, it is more efficient to incrementally update the MV instead of recomputation. There has been substantial work in deriving incremental updates (incremental maintenance) for different classes of MVs and optimizing their execution [7,18]. For frequently changing tables even incremental maintenance can be expensive since every update to the base data requires updating all the dependent views. In many important applications, such as summary statistics from user activity logs, new records arrive at a fast rate and data are often distributed across multiple machines making eager incremental maintenance infeasible. As a result, in production environments, it is common to defer view maintenance [7,9,34] so that updates can be batched together to amortize overheads and can be scheduled at times of low system utilization (e.g., nightly).

While deferring maintenance has benefits, a disadvantage is that MVs become increasingly stale between maintenance periods. As

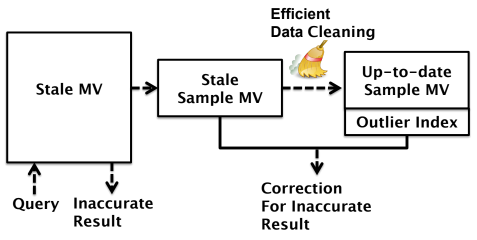


Figure 1: Deferred maintenance can lead to stale MVs which have incorrect, missing, and superfluous rows. In SVC, we pose this as a data cleaning problem and show that we can use a sample of clean (up-to-date) rows from an MV to correct inaccurate query results on the stale view with improved robustness using an outlier index.

a result, queries using those MVs can return incorrect answers. The problem of stale MVs parallels the problem of dirty data studied in data cleaning [28]. As with dirty data, a stale MV has incorrect, missing, or superfluous rows. In this work, we explore how answering queries on a stale MV can be formalized as a data cleaning problem.

Data cleaning has been studied extensively in the literature (e.g., see Rahm and Do for a survey [28]) but increasing data volumes have led to development of new, efficient sampling-based approaches for coping with dirty data. In our prior work, we developed the SampleClean framework for scalable aggregate query processing on dirty data [30]. Since data cleaning is often expensive, we proposed cleaning a sample of data using this sample to improve the results of aggregate queries on the full dataset. Since stale MVs are dirty data, an approach similar to SampleClean raises a new possibility, namely, we can use a sample of "clean" rows in the MV to return more accurate query results.

In this paper, we propose Stale View Cleaning (SVC), which uses data cleaning to stale MVs. SVC (Figure 1) provides a framework that efficiently cleans a sample of rows from a stale MV resulting in a uniform sample of "clean" (up-to-date) rows. After cleaning, we use the clean sample of rows to estimate a result for an aggregate query on the view. The estimates from this procedure, while approximate, are up-to-date in the sense that they reflect the most recent data. The approximation error due to sampling is more manageable than staleness because: (1) the uniformity of sampling allows us to apply theory from statistics such as the Central Limit Theorem to give tight bounds on approximate results, and (2) the approximate error is parametrized by the sample size which the user can control trading off accuracy for computation. SVC is complementary to existing deferred maintenance approaches. When the MVs become stale between maintenance cycles, we apply SVC for a far smaller cost than having to maintain the entire view but still get approximate, up-to-date answers.

To summarize, our contributions are as follows: (1) we formalize maintenance of a sample MV as a data cleaning operation on the sample, (2) we propose an optimization technique that materializes the clean sample efficiently while preserving correctness, (3) we derive a query processing approach to answer aggregate queries accurately using the clean sample, (4) we propose an outlier index to reduce sensitivity to skewed datasets, and (5) we evaluate our approach on real and synthetic datasets confirming that indeed sampling can reduce view maintenance time while providing accurate query results.

The paper is organized as follows: In Section 2, we give the necessary background for our work. Next, in Section 3, we formalize the problem. In Sections 4 and 5, we describe the sampling and query processing of our technique. In Section 6, we describe the outlier indexing framework. Then, in Section 7, we evaluate our approach. Finally, we discuss Related Work in Section 8. In Section 9, we discuss the limitations of our approach, and we present our Conclusions in Section 10.

2. BACKGROUND

2.1 Motivation And Example

There has been significant research on fast MV maintenance algorithms, most recently DBToaster [18]. However, even for these optimized systems, some materialized views are computationally difficult to maintain. In their evaluation of DBToaster, Koch et al. [18] reported a three order of magnitude variation in maintenance throughput over the 22 TPCH queries defined as MVs. Furthermore, in real deployments, it is common to use the same infrastructure to maintain multiple materialized views (along with other analytics tasks) adding further contention to computational resources and reducing overall maintenance throughput. SVC is complementary to existing maintenance algorithms, whether recomputation, IVM, or higher order methods like DBToaster, provided they are specified in standard relational algebra. Through the use of sampling, we provide the user another tool to flexibly manage and schedule maintenance of MVs for aggregate analytics.

Log Analysis Example: Suppose we are a video streaming company analyzing user engagement. Our database consists of two tables Log and Video, with the following schema:

```
Log(sessionId, videoId, responseTime, userAgent)
Video(videoId, ownerId, lang, duration)
```

The Log table stores each visit to a specific video with primary key (`sessionId`), a foreign-key to the Video table (`videoId`), latency of the visit (`responseTime`), and the browser used to access the video (`userAgent`). The Video stores each video with the primary key (`videoId`), a number identifying the owner of the view (`ownerId`), language (`lang`), and the video duration (`duration`).

For our analysis, we are interested in finding aggregate statistics on visits, such as the average visits per video and the total number of visits for each language, predicated on different subsets of owners. To avoid repeatedly performing the join and visit aggregation, we could define the following MV:

```
CREATE VIEW visitView
AS SELECT videoId, ownerId, language, duration
count(1) as visitCount
FROM Log, Video
WHERE Log.videoId = Video.videoId
GROUP BY videoId, ownerId, language, duration
```

Our queries of interest are aggregates on this view, are of the form:

```
SELECT agg(visitCount) FROM visitView
WHERE Condition(*)
```

As Log table grows, this MV becomes stale, and let us denote the insertions and deletions to the table as:

```
LogIns(sessionId, videoId, responseTime, userAgent)
LogDels(sessionId, videoId, responseTime, userAgent)
```

Intuitively, if we take a uniform sample of the rows in `visitView` and do sufficient computation to materialize updates from `LogIns` and `LogDels` to just those rows, we can reduce the maintenance time of the view. If we are intelligent about restricting the computation, this can reduce the number of Log records to aggregate and to join with the `Video` table. However, from this uniform sample, we can still estimate our aggregate queries of interest, trading off maintenance throughput for accuracy.

2.2 SampleClean [30]

In this paper, we formalize the intuition from our log analysis example. To do this, we leverage theory developed for query processing on dirty data. SampleClean is a framework for scalable aggregate query processing on dirty data. Traditionally, data cleaning has explored expensive, up-front cleaning of entire datasets for increased query accuracy. Those who were unwilling to pay the full cleaning cost avoided data cleaning altogether. We proposed SampleClean to add an additional trade-off to this design space by using sampling, i.e. bounded results for aggregate queries when only a sample of data is cleaned. The problem of high computational costs for accurate results mirrors the challenge faced in the MV setting with the tradeoff between immediate maintenance (expensive and up-to-date) and deferred maintenance (inexpensive and stale). Thus, we explore how samples of "clean" (up-to-date) data can be used for improved query processing on MVs without incurring the full cost of maintenance.

However, the metaphor of stale MVs as a Sample-and-Clean problem only takes so far. In SampleClean, we modeled data cleaning as a row-by-row transformation of relation with an unknown expensive cost. This transformation was a user-specified "black box" that operated on each row, and we applied this to a sample of dirty data. In the MV setting, we found that the staleness cleaning problem interacts with sampling in complex ways which results in missing and superfluous rows. Furthermore, MV maintenance does not necessarily commute with sampling, nor is sampling guaranteed to save on maintenance cost and it requires analysis to best optimize the sampling. We also greatly expand the query processing scope of SampleClean beyond `sum`, `count`, and `avg` queries as studied in that work. This requires new analytic tools such as statistical bootstrap estimation to calculate bounds.

3. FRAMEWORK OVERVIEW

We first define notation, terminology, and the problem setting that we address in this work. Then, we formalize the two main problems that SVC addresses: (1) cleaning the staleness errors in a sample of a MV and (2) answering an aggregate query with a clean sample.

3.1 Notation and Definitions

In SVC, we explore the problem of approximate aggregate query processing on stale materialized views using a data cleaning approach. We assume that these materialized views are periodically maintained and thus are stale in between maintenance periods. The focus of this paper is analytic workloads where the typical query is a group by aggregate on relatively large views. SVC provides a framework for increased query accuracy for a flexible additional maintenance cost that can scale with system constraints.

Materialized View: Let $\mathcal{D}$ be a database which is a collection of relations $\{R_i\}$. A *materialized view* S is the result of applying a *view definition* to $\mathcal{D}$. View definitions are composed of standard relational algebra expressions: Select (σ_ϕ), Project (Π), Join ($\bowtie$), Aggregation (γ), Union ($\cup$), and Intersection ($\cap$). We use the following parametrized notation for joins, aggregations and generalized projections:

- $\Pi_{a_1,a_2,...,a_k}(R)$: Generalized projection selects attributes $\{a_1, a_2, ..., a_k\}$ from R, allowing for new columns that are arithmetic transformations of attributes (e.g., $a_1 + a_2$).
- $\bowtie_{\phi(r1,r2)} (R_1, R_2)$: Join selects all tuples in $R_1 \times R_2$ that satisfy $\phi(r_1, r_2)$. We use $\bowtie$ to denote all types of joins even extended outer joins such as $⟗, ⟕, ⟖$.
- $\gamma_{f,A}(R)$: Apply the aggregate function f to the relation R grouped by the distinct values of A, where A is a subset of the attributes. The DISTINCT operation can be considered as a special case of the Aggregation operation.

The composition of the unary and binary relational expressions can be represented as a tree, which is called the *expression tree*. At the leaves of the tree are all of the *base relations* for a view. Each node of the tree is the result of applying one of the above relational expressions to a relation. To avoid ambiguity, we refer to tuples of the base relations as *records* and tuples of derived relations as *rows*.

Primary Key: We assume that each of the base relations has a *primary key*. If this is not the case, we can always add an extra column that assigns an increasing sequence of integers to each record. For the defined relational expressions, every row in a materialized view can be also be given a primary key [12,32], which we will describe in Section 4. This primary key is formally a subset of attributes $u \subseteq \{a_1, a_2, ..., a_k\}$ such that all $s \in S(u)$ are unique. We denote the entire row for that primary key as a selection $\sigma_u(S)$.

Staleness: For each relation R_i there is a set of insertions ΔR_i (modeled as a relation) and a set of deletions ∇R_i. An "update" to R_i can be modeled as a deletion and then an insertion. We refer to the set of insertion and deletion relations as "delta relations" denoted by $\partial \mathcal{D}$:

$$\partial \mathcal{D} = \{\Delta R_1, ..., \Delta R_k\} \cup \{\nabla R_1, ..., \nabla R_k\}$$

A view S is considered *stale* when there exist insertions or deletions to any of its base relations. This means that at least one of the delta relations in $\partial \mathcal{D}$ is non-empty.

Maintenance: There may be multiple ways (e.g., incremental maintenance or recomputation) to maintain a view S, and we denote the up-to-date view as S'. We formalize the procedure to maintain the view as a *maintenance strategy* $\mathcal{M}$. A maintenance strategy is a relational expression the execution of which will return S'. It is a function of the database $\mathcal{D}$, the stale view S, and all the insertion and deletion relations $\partial \mathcal{D}$. In this work, we consider maintenance strategies composed of the same relational expressions as materialized views described above.

$$S' = \mathcal{M}(S, \mathcal{D}, \partial D)$$

Staleness as Data Error: The consequences of staleness are incorrect, missing, and superfluous rows. Formally, for a stale view S with primary key u and an up-to-date view S':

- **Incorrect:** Incorrect row errors are the set of rows (identified by the primary key) that are updated in S':
$$\{\forall u \in S : (\exists u \in S' \wedge (\sigma_u(S) \neq \sigma_u(S')))\}$$
- **Missing:** Missing row errors are the set of rows (identified by the primary key) that exist in the up-to-date view but not in the stale view:
$$\{\forall u \in S' : \nexists u \in S\}$$
- **Superfluous:** Superfluous row errors are the set of rows (identified by the primary key) that exist in the stale view but not in the up-to-date view :
$$\{\forall u \in S : \nexists u \in S'\}$$

Uniform Random Sampling: We define a sampling ratio $m \in [0, 1]$ and for each row in a view S, we include it into a sample with probability m. We use the "hat" notation (e.g., $\widehat{S}, \widehat{\mathcal{M}(\cdot)}$) to denote sampled relations and sampled relational expressions. We say the relation $\hat{S}$ is a *uniform sample* of S if

$$(1)\ \forall s \in \hat{S} : s \in S; \quad (2)\ Pr(s_1 \in \hat{S}) = Pr(s_2 \in \hat{S}) = m$$

We say a sample is *clean* if an only if it is a uniform random sample of the up-to-date view S'.

EXAMPLE 1. *In this example, we summarize all of the key concepts and terminology pertaining to materialized views, stale data error, and maintenance strategies. Our example view, visitView, joins the Log table with the Video table and counts the visits for each video grouped by videoId, language, ownerId, and duration. Since there is a foreign key relationship between the relations, this is just a visit count for each unique video with additional attributes. The primary keys of the base relations are: sessionId for Log and videoId for Video.*

If new records have been added to (or deleted from) the Log table the visitView is considered stale. Incorrect rows in the view are videos for which the visitCount is incorrect, missing rows are videos that had not yet been viewed once at the time of materialization, and superfluous rows are videos whose Log records have all been deleted.

The maintenance strategy for this view is as follows (for simplicity consider the case with no deletions):

1. *Create a "delta view" by applying the view definition to LogIns. That is, calculate the visit count per video, language, and ownerId on the new logs.*
2. *Take the full outer join of the "delta view" with the stale view visitView (equality on videoId, language, ownerId).*
3. *Apply the generalized projection operator to add the visitCount in the delta view to each of the rows in visitView and treating the outer join NULL for either visitView or the "delta view" as 0.*

3.2 SVC Workflow

Formally, the workflow of SVC is:

1. We are given a view S.
2. $\mathcal{M}$ defines the maintenance strategy that updates S at each maintenance period.
3. The view S is stale between periodic maintenance, and the up-to-date view should be S'.
4. *(Problem 1: Stale Sample View Cleaning)* We find an expression $\mathcal{C}$ derived from $\mathcal{M}$ that cleans a uniform random sample of the stale view $\widehat{S}$ to produce a "clean" sample of the up-to-date view $\widehat{S'}$.
5. *(Problem 2. Query Result Estimation)* For an aggregate query q, we use $\widehat{S'}$ to estimate the up-to-date result.
6. We optionally maintain an index of outliers o for improved estimation in skewed data.

Stale Sample View Cleaning: The first problem addressed in this paper is how to clean a sample of the stale materialized view.

PROBLEM 1 (STALE SAMPLE VIEW CLEANING). *We are given a stale view S, a sample of this stale view $\widehat{S}$ with ratio m, the maintenance strategy $\mathcal{M}$, the base relations $\mathcal{D}$, and the insertion and deletion relations $\partial\mathcal{D}$. We want to find a relational expression $\mathcal{C}$ such that:*

$$\widehat{S}' = \mathcal{C}(\widehat{S}, \mathcal{D}, \partial\mathcal{D})$$

Where $\widehat{S}'$ is a sample of the up-to-date view with ratio m.

Query Result Estimation: The second problem addressed in this paper is query result estimation.

PROBLEM 2 (QUERY RESULT ESTIMATION). *Let q be an aggregate query of the following form* [1]*:*

[1]For simplicity, we exclude the group by clause for all queries in the paper, as it can be modeled as part of the Condition.

```
SELECT agg(a) FROM View WHERE Condition(A);
```

If the view S is stale, then the result will be incorrect by some value c:

$$q(S') = q(S) + c$$

Our objective is to find an estimator f such that:

$$q(S') \approx f(q(S), \widehat{S}, \widehat{S}')$$

EXAMPLE 2. *Suppose a user wants to know how many videos have received more than 100 views.*

```
SELECT COUNT(1) FROM visitView WHERE visitCount > 100;
```

Let us suppose the user runs the query and result is 45. However, there have now been new records inserted into the Log table making this result stale (for clarity no changes to Video or deletions). First, we take a sample of visitView and suppose this sample is a 5% sample. In Stale Sample View Cleaning (Problem 1), we calculate an expression $\mathcal{C}$ that takes the maintenance strategy $\mathcal{M}$ described in Example 1, takes the database $\mathcal{D}$ (Log and Video), and the delta relation $\partial\mathcal{D}$ (LogIns) as input and updates the visit counts for the sample. In Query Result Estimation (Problem 2), we take the result of running $\mathcal{C}$, which is a sample of the up-to-date visit counts, and estimate our above query. Suppose 2 videos have changed their counts from less than 100 to greater than 100. From this 5% sample, we extrapolate that 40 new videos throughout the view should now be included in the count. This means that we should correct the old result by 40 resulting in the estimate of $45 + 40 = 85$.

4. EFFICIENTLY CLEANING A SAMPLE

In this section, we describe how to find a relational expression $\mathcal{C}$ derived from the maintenance strategy $\mathcal{M}$ that efficiently cleans a sample of a stale view $\widehat{S}$ and produces a sample of the up-to-date view $\widehat{S}'$.

4.1 Challenges

To illustrate the challenges in deriving $\mathcal{C}$, we present two naive solutions to this problem that will not work. First, the maintenance strategy $\mathcal{M}$ can be thought of as a data cleaning procedure to clean these errors, since applying the strategy to a stale view S and the delta relations $\partial\mathcal{D}$ returns an up-to-date view S' without data error. We could trivially apply $\mathcal{M}$ to the entire stale view S and update it to S', and then sample. While the result is correct according to our problem formulation, it does not save us on any computation for maintenance. We want to restrict our cleaning to a sample requires so that it requires less computation by avoiding materialization of up-to-date rows outside of the sample. However, the alternative solution is also flawed. For example, we could just apply $\mathcal{M}$ to the stale sample $\widehat{S}$ and a sample of the delta relations $\widehat{\partial\mathcal{D}}$. For general maintenance strategies, $\mathcal{M}$ does not commute with sampling.

4.2 Provenance

We explore the commutativity problem in more detail. Consider the case of maintaining a view that is a group by aggregate:

```
SELECT videoId, count(1) FROM Log
GROUP BY videoId
```

The resulting view has one row for every distinct `videoId`. We want to materialize a sample of S', that is a sample of videos with up-to-date counts. If we randomly sample the delta relations $\widehat{\partial\mathcal{D}}$, we get a subset of records from LogIns. The problem is that if we propagate the updates based on $\widehat{\partial\mathcal{D}}$ to $\widehat{S}$ it is not guaranteed that every count is completely up-to-date since we may not sample all the records for some groups. This is because to achieve a sample of S', we need to ensure that for each $s \in S'$ all contributing rows in subexpressions to s are also sampled.

This is a problem of row provenance [12]. Provenance, also termed lineage, has been an important tool in the analysis of materialized views [12] and in approximate query processing [32].

DEFINITION 1 (PROVENANCE). *Let r be a row in relation R, let R be derived from some other relation $R = exp(U)$ where $exp(\cdot)$ be a relational expression composed of the expressions defined in Section 3.1. The provenance of row r with respect to U is $p_U(r)$. This defined as the set of rows in U such that for an update to any row $u \notin p_U(r)$, it guarantees that r is unchanged.*

4.3 Primary Keys

For the relational expressions defined in the previous sections, this provenance is well defined and can be tracked using primary key rules that are enforced on each subexpression [12]. So each row will have a designated primary key that will propagate to the next level of the relational tree. Formally, we recursively define a set of primary keys for all nodes in the expression tree:

DEFINITION 2 (PRIMARY KEY GENERATION). *For every relational expression R, we define the primary key attribute(s) of every expression to be:*

- *Base Case: All relations (leaves) must have an attribute p which is designated as a primary key. That uniquely identifies rows.*
- *$\sigma_\phi(R)$: Primary key of the result is the primary key of R*
- *$\Pi_{(a_1,\ldots,a_k)}(R)$: Primary key of the result is the primary key of R. The primary key must always be included in the projection.*
- *$\bowtie_{\phi(r1,r2)} (R_1, R_2)$: The primary key of the result is the tuple of the primary keys of R_1 and R_2.*
- *$\gamma_{f,A}(R)$: The primary key of the result is the group by key A (which may be a set of attributes).*
- *$R_1 \cup R_2$: Primary key of the result is the union of the primary keys of R_1 and R_2*
- *$R_1 \cap R_2$: Primary key of the result is the intersection of the primary keys of R_1 and R_2*
- *$R_1 - R_2$: Primary key of the result is the primary key of R_1*

For every node at the expression tree, these keys are guaranteed to uniquely identify a row.

These rules define a constructive definition that can always be applied for our defined relational expressions. We only have to ensure that any projection operation Π includes the operand's primary key, and enforcing this condition defines an equivalent materialized view. As we will subsequently see, this primary key definition allows us to efficiently sample the relational expression. For the relational expressions defined in the previous section, this method will always work. However, in the future, if we expand the allowed set of relational expressions there are some limitations which we defer to future work.

EXAMPLE 3. *A variant of our running example view that does not have a primary key is:*

```
CREATE VIEW visitView
AS SELECT count(1) as visitCount
FROM Log, Video
WHERE Log.videoId = Video.videoId
GROUP BY videoId, ownerId, language, duration
```

To ensure that this view has a primary key for all subexpression, we add the primary key to the projection:

```
CREATE VIEW visitView
AS SELECT videoId, ownerId, language, duration,
count(1) as visitCount
FROM Log, Video
WHERE Log.videoId = Video.videoId
GROUP BY videoId, ownerId, language, duration
```

These two views are equivalent, as any query that can be answered with the previous view can be answered with the new view with the additional attributes.

*Suppose there is a base relation, such as **Log**, that is missing a primary key (sessionId)*[2]*. We can add this attribute by generating an increasing sequence of integers for each record in **Log**. Then, we can apply the rules above to propagate this key up the expression tree.*

4.4 Hashing Operator

These primary keys define the provenance of a row r, which allows us to easily determine the set of rows in subexpressions that contribute to r:

PROPOSITION 1 (PRIMARY KEY PROVENANCE). *Let R and U be relations as defined in Definition 1. Let A_R be the primary key set of R and A_U be the primary key set of U. Define $r(A_R)$ as the primary key sets values for the row r. $p_U(r)$ is defined as follows: (1) if $A_R \subseteq A_U$ then $\{u \in U : u(A_R) = r(A_R)\}$, (2) if $A_R \not\subseteq A_U$ then return U.*

We now explore how we can design a sampling technique to guarantee that all of the rows in Proposition 1 are sampled if r is sampled. If we have a deterministic way of mapping a primary key defined in the previous subsection to Boolean true or false, we can ensure that all contributing rows are also sampled. To achieve this we use a hashing procedure. Let us denote the hashing operator $\eta_{a,m}(R)$. For all tuples in R, this operator applies a hash function whose range is $[0, 1]$ to primary key a (which may be a set) and selects those records with hash value less than or equal to m. If the hash function is sufficiently uniform, then the condition $h(a) \leq m$ is true for close to a fraction m of the rows.

We push down the hashing operator through the query tree. The further that we can push η down the expression tree, the more operators can benefit from the sampling. However, it is important to note that for some of the expressions, notably joins, the push down rules are more complex. It turns out in general we cannot push down even a deterministic sample through those expressions. We formalize the push down rules below:

DEFINITION 3 (HASH PUSHDOWN). *Let a be a primary key of a materialized view. The following rules can be applied to push $\eta_{a,m}(R)$ down the expression tree of the maintenance strategy.*

- *$\sigma_\phi(R)$: Push η through the expression.*
- *$\Pi_{(a_1,...,a_k)}(R)$: Push η through if a is in the projection.*
- *$\bowtie_{\phi(r1,r2)} (R_1, R_2)$: Blocks η in general. There are special cases below where push down is possible.*
- *$\gamma_{f,A}(R)$: Push η through if a is in the group by clause A.*
- *$R_1 \cup R_2$: Push η through to both R_1 and R_2*
- *$R_1 \cap R_2$: Push η through to both R_1 and R_2*
- *$R_1 - R_2$: Push η through to both R_1 and R_2*

In special cases, we can push the hashing operator down through joins. Given the hash function $\eta_{a,m}(R)$:

Equality Join: If the join is an equality join and a is one of the attributes in the equality join condition $R_1.a = R_2.b$, then η can be pushed down to both R_1 and R_2. On R_1 the pushed down operator is $\eta_{a,m}(R_1)$ and on R_2 the operator is $\eta_{b,m}(R_2)$. This case often happens near the top of maintenance strategy expression tree where there is a equality outer join on the primary key of the stale view and a "delta view".

Foreign Key Join: If we have a join with two foreign-key relations R_1 (fact table with foreign key a) and R_2 (dimension table with primary key $b \subseteq a$) and we are sampling the key a, then we can push the sampling down to R_1. This is because we are guaranteed that for every $r_1 \in R_1$ there is only one $r_2 \in R_2$. This case happens in our running example. If we sample the view on the primary key (`videoId`, `ownerId`, `language`, `duration`), since each video has only one owner, language and duration, we can push down the sampling of `videoId` to the **Log** relation and the **LogIns** table.

The result of this hash operator pushdown on $\mathcal{M}$ is the cleaning expression $\mathcal{C}$. When applied to a stale sample of a view $\widehat{S}$, the database $\mathcal{D}$, and the delta relations $\partial\mathcal{D}$, it produces an up-to-date sample with sampling ratio m:

$$\widehat{S}' = \mathcal{C}(\widehat{S}, \mathcal{D}, \partial\mathcal{D})$$

Thus, it addresses Problem 1 from the previous section.

EXAMPLE 4. *We illustrate our proposed approach on our example view `visitView` (Figure 2). The primary key for the view is the tuple (`videoId`, `ownerId`, `language`, `duration`) making that the primary key of the MV. We start by applying the hashing operator to this key. The next operator we see in the expression tree is a projection that increments the `visitCount` in the view, and this allows for push down since primary key is in the projection. The second expression is a hash of the equality join key which merges the aggregate from the "delta view" to the old view allowing us to push down on both branches of the tree using our special case for equality joins. On the left side, we reach the stale view so we stop. On the right side, we reach the aggregate query (count) and since the primary key is in group by clause, we can push down the sampling. Then, we reach another point where we hash the equality join key allowing us to push down the sampling to the relations **LogIns** and **Video**.*

4.5 Corresponding Samples

We started with a uniform random sample $\widehat{S}$ of the stale view S. The hash push down allows us to efficiently materialize the sample $\widehat{S}'$. $\widehat{S}'$ is a uniform random sample of the up-to-date view S. While both of these samples are uniform random samples of their respective relations, the two samples are correlated since $\widehat{S}'$ is generated by cleaning $\widehat{S}$. In particular, our hashing technique ensures that the primary keys in $\widehat{S}'$ depend on the primary keys in $\widehat{S}$. Statistically, this positively correlates the query result $q(\widehat{S}')$ and $q(\widehat{S})$. We will see how this property can be leveraged to improve query estimation accuracy (Section 5.1).

PROPERTY 1 (CORRESPONDENCE). *Suppose $\widehat{S}'$ and $\widehat{S}$ are uniform samples of S' and S, respectively. Let u denote the primary key. We say $\widehat{S}'$ and $\widehat{S}$ correspond if and only if:*

- *Uniformity: $\widehat{S}'$ and $\widehat{S}$ are uniform random samples of S' and S respectively with a sampling ratio of m*
- *Removal of Superfluous Rows: $D = \{\forall u \in \widehat{S} \wedge u \notin S'\}$, $D \cap \widehat{S}' = \emptyset$*
- *Sampling of Missing Rows: $I = \{\forall u \notin S : u \notin S\}$,*
$$\mathbb{E}(\mid I \cup \widehat{S}' \mid) = m \mid I \mid$$
- *Key Preservation: For all $u \in \widehat{S}$ and not in D or I, $u \in \widehat{S}'$.*

5. QUERY RESULT ESTIMATION

SVC returns two corresponding samples, $\widehat{S}$ and $\widehat{S}'$. $\widehat{S}$ is a "dirty" sample (sample of the stale view) and $\widehat{S}'$ is a "clean" sample (sample of the up-to-date view). In this section, we first discuss how to estimate query results using the two corresponding samples. Then, we discuss the bounds and guarantees on different classes of aggregate queries.

[2]It does not make sense for Video to be missing a primary key in our running example due to the foreign key relationship

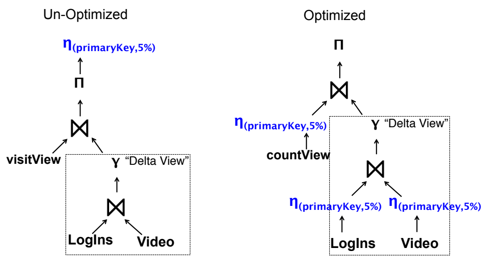


Figure 2: Applying the rules described in Section 4.4, we illustrate how to optimize the sampling of our example maintenance strategy.

5.1 Result Estimation

Suppose, we have an aggregate query q of the following form:

```
q(View) := SELECT f(attr) FROM View WHERE cond(*)
```

We quantify the staleness c of the aggregate query result as the difference between the query applied to the stale view S compared to the up-to-date view S':

$$q(S') = q(S) + c$$

The objective of this work is to estimate $q(S')$. In the Approximate Query Processing (AQP) literature, sample-based estimates have been well studied [3,26]. This inspires our first estimation algorithm, SVC+AQP, which uses SVC to materialize a sample view and an AQP-style result estimation technique.

SVC+AQP: Given a clean sample view $\widehat{S}'$, the query q, and a scaling factor s. We apply the query to the sample and scale it by s:

$$q(S') \approx s \cdot q(\widehat{S}')$$

For example, for the `sum` and `count` the scaling factor is $\frac{1}{m}$. For the `avg` the scaling factor is 1. Refer to [3,26] for a detailed discussion on the scaling factors.

SVC+AQP returns what we call a direct estimate of $q(S')$. We could, however, try to estimate c instead. Since we have the stale view S, we could run the query q on the full stale view and estimate the difference c using the samples $\widehat{S}$ and $\widehat{S}'$. We call this approach SVC+CORR, which represents calculating a correction to $q(S)$ instead of a direct estimate.

SVC+CORR: Given a clean sample $\widehat{S}'$, its corresponding dirty sample $\widehat{S}$, a query q, and a scaling factor s:

1. Apply SVC+AQP to $\widehat{S}'$:
$$r_{est\_fresh} = s \cdot q(\widehat{S}')$$
2. Apply SVC+AQP to $\widehat{S}$:
$$r_{est\_stale} = s \cdot q(\widehat{S})$$
3. Apply q to the full stale view:
$$r_{stale} = q(S)$$
4. Take the difference between (1) and (2) and add it to (3):
$$q(S') \approx r_{stale} + (r_{est\_fresh} - r_{est\_stale})$$

A commonly studied property in the AQP literature is unbiasedness. An unbiased result estimate means that average value of the estimate over all possible samples is $q(S')$. We can prove that if SVC+AQP is unbiased (there is an AQP method that gives an unbiased result) then SVC+CORR also gives unbiased results.

LEMMA 1. *If there exists an unbiased sample estimator for q(S') then there exists an unbiased sample estimator for c.*

PROOF SKETCH. Suppose, we have an unbiased sample estimator e_q of q. Then, it follows that

$$\mathbb{E}\big[e_q(\hat{S}')\big] = q(S')$$

If we substitute in this expression: $c = \mathbb{E}\big[e_q(\hat{S}')\big] - q(s)$ Applying the linearity of expectation:

$$c = \mathbb{E}\big[e_q(\hat{S}') - q(s)\big]$$

SQL Query	Estimator Family	Unbiased	Estimate Variance
`avg`, `sum`, `count`	Sample Mean	Yes	Optimal
`std`, `var`	Sample Variance	Yes	Optimal
`median`, `percentile`	Sample Ranking	Bounded	Suboptimal
`max`, `min`	Sample Extrema	Unbounded	Suboptimal

Table 1: SQL queries and the properties of their statistical estimation family.

□

Some queries do not have unbiased sample estimators, but the bias of their sample estimators can be bounded. Example queries include: `median`, `percentile`. A corollary to the previous lemma, is that if we can bound the bias for our estimator then we can achieve a bounded bias for c as well.

EXAMPLE 5. *We can formalize our earlier example query in Section 2 in terms of SVC+CORR and SVC+AQP. Let us suppose the initial query result is* 45. *There now have been new log records inserted into the Log table making the old result stale, and suppose we are working with a sampling ratio of 5%. For SVC+AQP, we count the number of videos in the sample that currently have counts greater than 100 and scale that result by 20. For SVC+CORR, suppose 2 videos have changed their counts from less than 100 to greater than 100. From this sample, we calculate how many new videos changed from less than 100 views to times greater than 100; let us suppose this answer is* 2. *We extrapolate that* 40 *new videos throughout the view should now be included in the count. This means that we should correct the old result by* 40 *resulting in the estimate of* $45 + 40 = 85$.

5.2 Estimate Accuracy

To analyze the estimate accuracy, we taxonomize common SQL aggregate queries into different *estimator families*. For example, `sum`, `count`, and `avg` can all be written as sample means. `sum` is the sample mean scaled by the relation size and `count` is the mean of the indicator function scaled by the relation size. There are some key properties of interest within different estimator families: unbiasedness, existence analtyic confidence intervals, and optimality. SVC+AQP and SVC+CORR inherit the properties of the estimator family.

Table 1 describes these families and their properties for common queries. Sample mean family of estimators (`sum`, `count`, and `avg`) has analytic solutions and has been the focus of other approximate query processing works [26,30], we analyze this family in detail. The general case can only be bounded empirically which is more challenging.

5.2.1 Confidence Intervals For Sample Means

Now we will discuss bounding our estimates in confidence intervals for `sum`, `count`, and `avg`, which can be estimated with "sample mean" estimators. Sample means for uniform random samples (also called sampling without replacement) converge to the population mean by the Central Limit Theorem (CLT). Let $\hat{\mu}$ be a sample mean calculated from k samples, σ^2 be the variance of the sample, and μ be the population mean. Then, the error in the estimate is normally distributed[3]:

$$(\mu - \hat{\mu}) \sim N(0, \frac{\sigma^2}{k})$$

Therefore, the confidence interval is given by:

$$\hat{\mu} \pm \gamma \sqrt{\frac{\sigma^2}{k}}$$

where γ is the Gaussian tail probability value (eg. 1.96 for 95%, 2.57 for 99%).

[3]We can include a finite population correction if our relation size is small $\frac{N-k}{N-1}$.

Next, we discuss how to calculate this confidence interval in SQL for SVC+AQP. The first step is a query rewriting step where we express the query on the view in terms of a case statement (1 if true, 0 if false) instead of a predicate. Let *attr* be the aggregate attribute and m be the sampling ratio. We define an intermediate result $trans$ which is a table of transformed rows with the first column the primary key and the second column defined in terms of a case statement and scaling. For sum:

```
trans(sample):= SELECT pk,
1.0/m · attr · cond(*) as trans_attr
FROM sample
```

For avg since there is no scaling we do not need to re-write:

```
trans(sample) := SELECT pk, attr as trans_attr
WHERE cond(*) FROM sample
```

For count:

```
trans(sample) := SELECT pk, 1.0/m · cond(*) as trans_attr
FROM sample
```

SVC+AQP: The confidence interval on this result is defined as:

```
SELECT γ·stdev(trans_attr)/sqrt(count(1)) FROM trans
```

To calculate the confidence intervals for SVC+CORR we have to look at the statistics of c the difference i.e. $c = q(S) - q(S')$ from a sample. For queries that can be estimated with a sample mean, we can use the associativity of addition and subtraction to rewrite this as:

$$c = q(S - S')$$

Where $-$ is the row-by-row difference between S and S'. The problem is that some rows in $\widehat{S}$ do not exist in $\widehat{S}'$ and vice versa. Thus, we have define the following null-handling semantics with a subtraction operator we call $\dot{-}$.

Definition 4 (Correspondence Subtract). *Given an aggregate query, and two corresponding relations R_1 and R_2 with the schema $(a_1, a_2, ...)$ where a_1 is the primary key for R_1 and R_2, and a_2 is the aggregation attribute for the query. $\dot{-}$ is defined as a projection of the full outer join on equality of $R_1.a_1 = R_2.a_1$:*

$$\Pi_{R_1.a_2 - R_2.a_2}(R_1 ⟗ R_2)$$

Null values $\emptyset$ are represented as zero.

Using this operator, we can define a new intermediate result $diff$:

$$diff := trans(\widehat{S}') \dot{-} trans(\widehat{S})$$

SVC+CORR: Then, as in SVC+AQP, we bound the result using the CLT:

```
SELECT γ·stdev(trans_attr)/sqrt(count(1)) FROM diff
```

5.2.2 AQP vs. CORR For Sample Means

In terms of these bounds, we can analyze how SVC+AQP compares to SVC+CORR for a fixed sample size k. SVC+AQP gives an estimate that is proportional to the variance of the clean sample view: $\frac{\sigma^2_{S'}}{k}$. SVC+CORR to the variance of the *differences*: $\frac{\sigma^2_c}{k}$. Since the change is the difference between the stale and up-to-date view, this can be rewritten as

$$\frac{\sigma^2_S + \sigma^2_{S'} - 2cov(S, S')}{k}$$

Therefore, a correction will have less variance when:

$$\sigma^2_S \leq 2cov(S, S')$$

As we saw in the previous section, correspondence correlates the samples. If the difference is small, i.e. S is nearly identical to S',then $cov(S, S') \approx \sigma^2_S$. This result also shows that there is a point when updates to the stale MV are significant enough that direct estimates are more accurate. When we cross the break-even point we can switch from using SVC+CORR to SVC+AQP. SVC+AQP does not depend on $cov(S, S')$ which is a measure of how much the data has changed. Thus, we guarantee an approximation error of at most $\frac{\sigma^2_{S'}}{k}$. In our experiments (Figure 5(b)), we evaluate this break even point empirically.

5.2.3 Selectivity For Sample Means

Let p be the selectivity of the query and k be the sample size; that is, a fraction p records from the relation satisfy the predicate. For these queries, we can model selectivity as a reduction of effective sample size $k \cdot p$ making the estimate variance: $O(\frac{1}{k*p})$. Thus, the confidence interval's size is scaled up by $\frac{1}{\sqrt{p}}$. Just like there is a tradeoff between accuracy and maintenance cost, for a fixed accuracy, there is also a tradeoff between answering more selective queries and maintenance cost.

5.2.4 Optimality For Sample Means

Optimality in unbiased estimation theory is defined in terms of the variance of the estimate [11].

Proposition 2. *An estimator is called a minimum variance unbiased estimator (MVUE) if it is unbiased and the variance of the estimate is less than or equal to that of any other unbiased estimate.*

A sampled relation R defines a discrete distribution. It is important to note that this distribution is different from the data generating distribution, since even if R has continuous valued attributes R still defines a discrete distribution. Our population is finite and we take a finite sample thus every sample takes on only a discrete set of values. In the general case, this distribution is only described by the set of all of its values (i.e no smaller parametrized representation). In this setting, the sample mean is an MVUE. In other words, if we make no assumptions about the underlying distribution of values in R, SVC+AQP and SVC+CORR are optimal for their respective estimates ($q(S')$ and c). Since they estimate different variables, even with optimality SVC+CORR might be more accurate than SVC+AQP and vice versa.

However, if we do know more about the distribution, this optimality is not true in general. The intuitive problem is that if there are a small number of parameters that completely describe the discrete distribution there might be a way to reconstruct the distribution from those parameters rather than estimating the mean. As a simple counter example, if we knew our attributes were exactly on a line, a sample size of two is sufficient to answer any aggregate query. However, even for many parametric distributions, the sample mean estimators are still MVUEs, eg. poisson, bernouilli, binomial, normal, exponential. It is often difficult and unknown in many cases to derive an MVUE other than a sample mean. Furthermore, the sample mean is unbiased for all distribution, but it is often the case that alternative MVUEs are biased when the data is not exactly from correct model family (such as our example of the line). Our approach is valid for any choice of estimator if one exists, even though we do the analysis for sample mean estimators and this is the setting in which that estimator is optimal.

5.2.5 General Estimators

The theory for bounding general estimators outside of the sample mean family is more complex. We may not get analytic confidence intervals on our results, nor is it guaranteed that our estimates are optimal. In AQP, the commonly used technique is called a statistical bootstrap [3] to empirically bound the results. In this approach, we repeatedly subsample with replacement from our sample and apply the query to the sample. This gives us a technique to bound SVC+AQP the details of which can be found in [2,3,32]. For SVC+CORR, we have to propose a variant of bootstrap to bound the estimate of c. In this variant, repeatedly estimate c from subsamples and build an empirical distribution for c.

SVC+CORR: To use bootstrap to find a 95% confidence interval:

1. Subsample $\widehat{S}'_{sub}$ and $\widehat{S}_{sub}$ with replacement from $\widehat{S}'$ and $\widehat{S}$ respectively
2. Apply SVC+AQP to $\widehat{S}'_{sub}$ and $\widehat{S}_{sub}$
3. Record the difference $s \cdot (q(\widehat{S}'_{sub}) - q(\widehat{S}_{sub}))$
4. Return to 1, for k iterations.
5. Return the $100 - \alpha\%$ and the $\alpha\%$ percentile of the distribution of results.

6. OUTLIER INDEXING

Sampling is known to be sensitive to outliers [5,8]. Power-laws and other long-tailed distributions are common in practice [8]. We address this problem using a technique called outlier indexing which has been applied in AQP [5]. The basic idea is that we create an index of outlier records (records whose attributes deviate from the mean value greatly) and ensure that these records are included in the sample. The intuition is that these records greatly increase the variance of the data. Furthermore, since they are likely rare the probability of sampling them is low leading to wildly varying estimates. However, as this has not been explored in the materialized view setting there are new challenges in using this index for improved result accuracy.

6.1 Indices on the Base Relations

In [5], the authors applied outlier indexing to improve the accuracy of AQP. We apply a similar technique, however, their problem setting is different in a few ways. First, in the AQP setting, queries are issued to base relations. In our problem, we issue queries to materialized views. We need to define how to propagate information from an outlier index on the base relation to a materialized view.

The first step is that the user selects an attribute of any base relation to index and specifies a threshold t and a size limit k. In a single pass of updates (without maintaining the view), the index is built storing references to the records with attributes greater than t. If the size limit is reached, the incoming record is compared to the smallest indexed record and if it is greater then we evict the smallest record. The same approach can be extended to attributes that have tails in both directions by making the threshold t a range, which takes the highest and the lowest values. However, in this section, we present the technique as a threshold for clarity.

There are many approaches to select a threshold. We can use prior information from the base table, a calculation which can be done in the background during the periodic maintenance cycles. If our size limit is k, for a given attribute we can select the the top-k records with that attributes. Then, we can use that top-k list to set a threshold for our index. Then, the attribute value of the lowest record becomes the threshold t. Alternatively, we can calculate the variance of the attribute and set the threshold to represent c standard deviations above the mean.

This threshold can be adaptively set at each maintenance period to include more or less outliers. The caveat is that the outlier index should not be too expensive to calculate nor should it be too large as it negates the performance benefits of sampling. The query processing approach that we propose in the following sub-sections is agnostic to how we choose this threshold. In fact, our approach allows us to incorporate any deterministic subset into our sample-based correction calculations.

6.2 Adding Outliers to the Sample

We ensure that any row in a materialized view that is derived from an indexed record is guaranteed to be in the sample. This problem is sort of an inverse to the efficient sampling problem studied in Section 4. We need to propagate the indices upwards through the expression tree.

We add the condition that the only eligible indices are ones on base relations that are being sampled (i.e., we can push the hash operator down to that relation). Therefore, in the same iteration as sampling, we can also test the index threshold and add records to the outlier index. We formalize the propagation property recursively. Every relation can have an outlier index which is a set of attributes and a set of records that exceed the threshold value on those attributes.

The main idea is to treat the indexed records as a sub-relation that gets propagated upwards with the maintenance strategy.

DEFINITION 5 (OUTLIER INDEX PUSHUP). *Define an outlier index to be a tuple of a set of indexed attributes, and a set of records (I, O). The outlier index propagates upwards with the following rules:*

- *Base Relations: Outlier indices on base relations are pushed up only if that relation is being sampled, i.e., if the sampling operator can be pushed down to that relation.*
- $\sigma_\phi(R)$*: Push up with a new outlier index and apply the selection to the outliers* $(I, \sigma_\phi(O))$
- $\Pi_{(a_1,...,a_k)}(R)$*: Push upwards with new outlier index* $(I \cap (a_1, ..., a_k), O)$.
- $\bowtie_{\phi(r1,r2)} (R_1, R_2)$*: Push upwards with new outlier index* $(I_1 \cup I_2, O_1 \bowtie O_2)$.
- $\gamma_{f,A}(R)$*: For group-by aggregates, we set I to be the aggregation attribute. For the outlier index, we do the following steps. (1) Apply the aggregation to the outlier index* $\gamma_{f,A}(O)$*, (2) for all distinct A in O select the row in* $\gamma_{f,A}(R)$ *with the same A, and (3) this selection is the new set of outliers O.*
- $R_1 \cup R_2$*: Push up with a new outlier index* $(I_1 \cap I_2, O_1 \cup O_2)$*. The set of index attributes is combined with an intersection to avoid missed outliers.*
- $R_1 \cap R_2$*: Push up with a new outlier index* $(I_1 \cap I_2, O_1 \cap O_2)$.
- $R_1 - R_2$*: Push up with a new outlier index* $(I_1 \cup I_2, O_1 - O_2)$.

For all outlier indices that can propagate to the view (i.e., the top of the tree), we get a final set O of records. Given these rules, O is, in fact, a subset of our materialized view S'. Thus, our query processing can take advantage of the theory described in the previous section to incorporate the set O into our results. We implement the outlier index as an additional attribute on our sample with a boolean flag true or false if it is an outlier indexed record. If a row is contained both in the sample and the outlier index, the outlier index takes precedence. This ensures that we do not double count the outliers.

6.3 Query Processing

For result estimation, we can think of our sample $\hat{S}'$ and our outlier index O as two distinct parts. Since $O \subset S'$, and we give membership in our outlier index precedence, our sample is actually a sample restricted to the set $(\widehat{S' - O})$. The outlier index has two uses: (1) we can query all the rows that correspond to outlier rows, and (2) we can improve the accuracy of our *aggregation* queries. To query the outlier rows, we can select all of the rows in the materialized view that are flagged as outliers, and these rows are guaranteed to be up-to-date.

For (2), we can also incorporate the outliers into our correction estimates. For a given query, let c_{reg} be the correction calculated on $(\widehat{S' - O})$ using the technique proposed in the previous section and adjusting the sampling ratio m to account for outliers removed from the sample. We can also apply the technique to the outlier set O since this set is deterministic the sampling ratio for this set is $m = 1$, and we call this result c_{out}. Let N be the count of records that satisfy the query's condition and l be the number of outliers that satisfy the condition. Then, we can merge these two corrections in the following way: $v = \frac{N-l}{N} c_{reg} + \frac{l}{N} c_{out}$.

For the queries in the previous section that are unbiased, this approach preserves unbiasedness. Since we are averaging two unbiased estimates c_{reg} and c_{out}, the linearity of the expectation operator preserves this property. Furthermore, since c_{out} is deterministic (and in fact its bias/variance is 0), c_{reg} and c_{out} are uncorrelated making the bounds described in the previous section applicable as well.

EXAMPLE 6. *Suppose, we want to use outlier indexing to process queries on* visitView. *We chose an attribute in the base data to index, for example* `duration`. *We can set an threshold of 1.5 hours (e.g, indexes all of the movies served by the system). This materializes the entire set of rows whose duration is longer than 1.5 hours.*

Returning to the example query from the previous section:

SELECT COUNT*(1)* ***FROM*** *visitView* ***WHERE*** *visitCount > 100;*

We run SVC+AQP and SVC+CORR on the set of rows with durations longer than 1.5 hours. Then we use the update rule in Section 6.3 to update the result based on the number of records in the index and the total size of the view.

7. RESULTS

We evaluate SVC first on a single node MySQL database to evaluate its accuracy, performance, and efficiency in a variety of materialized view scenarios. Then, we evaluate the outlier indexing approach in terms of improved query accuracy and also evaluate the overhead associated with using the index. After evaluation on the benchmark, we present an application of server log analysis with a dataset from a video streaming company, Conviva. In this application, we look at the real query workload of the company and materialize views that could improve performance of these queries.

7.1 Experimental Setup

7.1.1 Single-node Experimental Setup

Our single node experiments are run on a r3.large Amazon EC2 node (2x Intel Xeon E5-2670, 15.25 GB Memory, and 32GB SSD Disk) with a MySQL version 5.6.15 database. These experiments evaluate views from 10GB TPCD and TPCD-Skew datasets. TPCD-Skew dataset [6] is based on the Transaction Processing Council's benchmark schema but is modified so that it generates a dataset with values drawn from a Zipfian distribution instead of uniformly. The Zipfian distribution [22] is a long-tailed distribution with a single parameter $z = \{1, 2, 3, 4\}$ which a larger value means a more extreme tail. $z = 1$ corresponds to the basic TPCD benchmark. The incremental maintenance algorithm used in our experiments is the "change-table" technique proposed in [16]. We implement incremental view maintenance with an "update...on duplicate key insert" command. We implement SVC's sampling operator with a linear hash written in C that is evoked in MySQL as a stored procedure. In all of the applications, the updates are kept in memory in a temporary table, and we discount this loading time from our experiments. We build an index on the primary keys of the view, the base data, but not on the updates. Below we describe the view definitions and the queries on the views:

Join View: In the TPCD specification, two tables receive insertions and updates: lineitem and orders. Out of 22 parametrized queries in the specification, 12 are group-by aggregates of the join of lineitem and orders (Q3, Q4, Q5, Q7, Q8, Q9, Q10, Q12, Q14, Q18, Q19, Q21). Therefore, we define a materialized view of the foreign-key join of lineitem and orders, and compare incremental view maintenance and SVC. We treat the 12 group-by aggregates as queries on the view.

Complex Views: Our goal is to demonstrate the applicability of SVC outside of simple materialized views that include nested queries and other more complex relational algebra. We take the TPCD schema and denormalize the database, and treat each of the 22 TPCD queries as views on this denormalized schema. The 22 TPCD queries are actually parametrized queries where parameters, such as the selectivity of the predicate, are randomly set by the TPCD qgen program. Therefore, we use the program to generate 10 random instances of each query and use each random instance as a materialized view. 10 out of the 22 sets of views can benefit from SVC. For the 12 excluded views, 3 were static (i.e, this means that there are no updates to the view based on the TPCD workload), and the remaining 9 views have a small cardinality not making them suitable for sampling.

For each of the views, we generated *queries on the views*. Since the outer queries of our views were group by aggregates, we picked a random attribute a from the group by clause and a random attribute b from aggregation. We use a to generate a predicate. For each attribute a, the domain is specified in the TPCD standard. We select a random subset of this domain ,e.g., if the attribute is country then the predicate can be countryCode > 50 and countryCode < 100. We generated 100 random `sum`, `avg`, and `count` queries for each view.

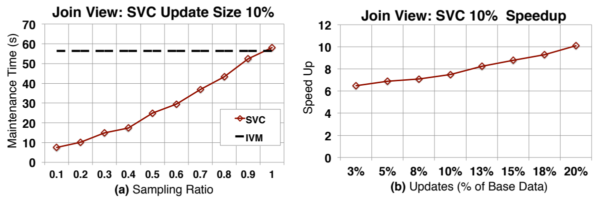


Figure 3: (a) On a 10GB view with 1GB of insertions and updates, we vary the sampling ratio and measure the maintenance time of SVC. (b) For a fixed sampling ratio of 10%, we vary the update size and plot the speedup compared to full incremental maintenance.

7.1.2 Distributed Experimental Setup

We evaluated our approach on a real dataset with Apache Spark 1.1.0 on a 10 node r3.large Amazon EC2 cluster. We evaluate SVC on a 1TB dataset of logs from a video streaming company, Conviva [1]. The dataset is a denormalized user activity log corresponding to video views and various metrics such as data transfer rates, and latencies. With this dataset, there was a corresponding dataset of analyst SQL queries on the log table. Using the dataset of analyst queries, we identified 8 common summary statistics-type queries that calculated engagement and error-diagnosis metrics for specific customers on a certain day. We generalized these queries by turning them into group-by queries over customers and dates; that is a view that calculates the metric for every customer on every day. We generated aggregate random queries over this view by taking either random time ranges or random subsets of customers.

7.1.3 Metrics and Evaluation

To illustrate how SVC gives the user access to this new tradeoff space, we will illustrate that SVC is more accurate than the stale query result (No Maintenance); but is less computationally intensive than full IVM. In our evaluation, we separate maintenance from query processing. We use the following notation to represent the different approaches:

No maintenance (Stale): The baseline for evaluation is not applying any maintenance to the materialized view.

Incremental View Maintenance (IVM): We apply incremental view maintenance (change-table based maintenance [16]) to the full view.

SVC+AQP: We maintain a sample of the materialized view using SVC and estimate the result with SVC+AQP.

SVC+CORR: We maintain a sample of the materialized view using SVC and process queries on the view using SVC+CORR.

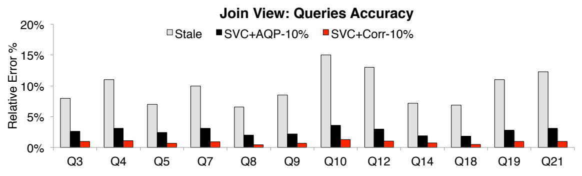


Figure 4: For a fixed sampling ratio of 10% and update size of 10% (1GB), we generate 100 of each TPCD parameterized queries and answer the queries using the stale materialized view, SVC+CORR, and SVC+AQP. We plot the median relative error for each query.

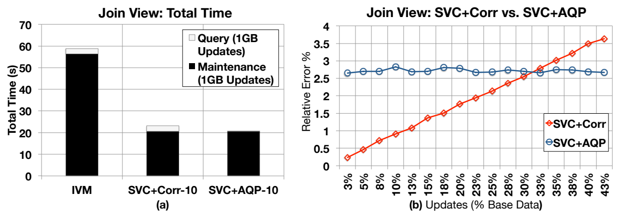


Figure 5: (a) For a fixed sampling ratio of 10% and update size of 10% (1GB), we measure the total time incremental maintenance + query time. (b) We vary the update rate to show that SVC+CORR is more accurate than SVC+AQP until a break even point.

Since SVC has a sampling parameter, we denote a sample size of $x\%$ as SVC+CORR-x or SVC+AQP-x, respectively. To evaluate accuracy and performance, we define the following metrics:

Relative Error: For a query result r and an incorrect result r', the relative error is $\frac{|r-r'|}{r}$. When a query has multiple results (a group-by query), then, unless otherwise noted, relative error is defined as the median over all the errors.

Maintenance Time: We define the maintenance time as the time needed to produce the up-to-date view for incremental view maintenance, and the time needed to produce the up-to-date sample in SVC.

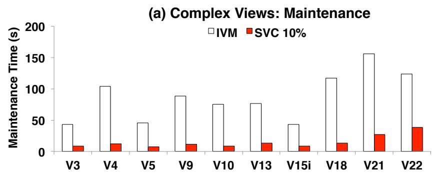
7.2 Join View

In our first experiment, we evaluate how SVC performs on a materialized view of the join of lineitem and orders. We generate a 10GB base TPCD dataset with skew $z = 1$, and derive the view from this dataset. We first generate 1GB (10% of the base data) of updates (insertions and updates to existing records), and vary the sample size.

Performance: Figure 3(a), shows the maintenance time of SVC as a function of sample size. With the bolded dashed line, we note the time for full IVM. For this materialized view, sampling allows for significant savings in maintenance time; albeit for approximate answers. While full incremental maintenance takes 56 seconds, SVC with a 10% sample can complete in 7.5 seconds.

The speedup for SVC-10 is 7.5x which is far from ideal on a 10% sample. In the next figure, Figure 3(b), we evaluate this speedup. We fix the sample size to 10% and plot the speedup of SVC compared to IVM while varying the size of the updates. On the x-axis is the update size as a percentage of the base data. For small update sizes, the speedup is smaller, 6.5x for a 2.5% (250MB) update size. As the update size gets larger, SVC becomes more efficient, since for a 20% update size (2GB), the speedup is 10.1x. The super-linearity is because this view is a join of lineitem and orders and we assume that there is not a join index on the updates. Since both tables are growing sampling reduces computation super-linearly.

Accuracy: At the same design point with a 10% sample, we evaluate the accuracy of SVC. In Figure 4, we answer TPCD queries with this view. The TPCD queries are group-by aggregates



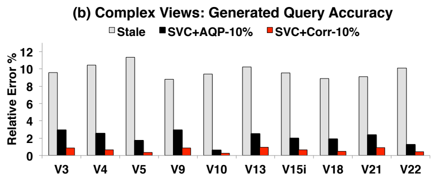


Figure 6: (a) For 1GB update size, we compare maintenance time and accuracy of SVC with a 10% sample on different views. V21 and V22 do not benefit as much from SVC due to nested query structures. (b) As before for a 10% sample size and 10% update size, SVC+CORR is more accurate than SVC+AQP and No Maintenance.

and we plot the median relative error for SVC+CORR, No Maintenance, and SVC+AQP. On average over all the queries, we found that SVC+CORR was 11.7x more accurate than the stale baseline, and 3.1x more accurate than applying SVC+AQP to the sample.

SVC+CORR vs. SVC+AQP: While more accurate, it is true that SVC+CORR correction technique moves some of the computation from maintenance to query execution. SVC+CORR calculates a correction to a query on the full materialized view. On top of the query time on the full view (as in IVM) there is additional time to calculate a correction from a sample. On the other hand SVC+AQP runs a query only on the sample of the view, We evaluate this overhead in Figure 5(a), where we compare the total maintenance time and query execution time. For a 10% sample SVC+CORR required 2.69 secs to execute a `sum` over the whole view, IVM required 2.45 secs, and SVC+AQP required 0.25 secs. However, when we compare this overhead to the savings in maintenance time it is small.

SVC+CORR is most accurate when the materialized view is less stale as predicted by our analysis. On the other hand SVC+AQP is more robust to the staleness and gives a consistent relative error. The error for SVC+CORR grows proportional to the staleness. In Figure 5(b), we explore which query processing technique should be used SVC+CORR or SVC+AQP. For a 10% sample, we find that SVC+CORR is more accurate until the update size is 32.5% of the base data.

7.3 Complex Views

In this experiment, we demonstrate the breadth of views supported by SVC by using the TPCD queries as materialized views.

Performance: Figure 6, shows the maintenance time for a 10% sample compared to the full view. This experiment illustrates how the view definitions plays a role in the efficiency of our approach. For the last two views, V21 and V22, we see that sampling does not lead to as large of speedup indicated in our previous experiments. This is because both of those views contain nested structures which block the pushdown of hashing. V21 contains a subquery in its predicate that does not involve the primary key, but still requires a scan of the base relation to evaluate. V22 contains a string transformation of a key blocking the push down. There might be a way to derive an equivalent expression with joins that could be sampled more efficiently, and we will explore this in future work. For the most part, these results are consistent with our previous experiments showing that SVC is faster than IVM and more accurate than

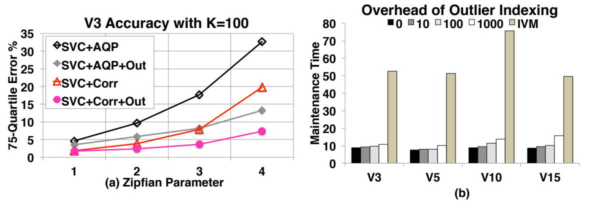


Figure 7: (a) For one view V3 and 1GB of updates, we plot the 75% quartile error with different techniques as we vary the skewness of the data. We find that SVC with an outlier index of size 100 is the most accurate. (b) While the outlier index adds an overhead this is small relative to the total maintenance time.

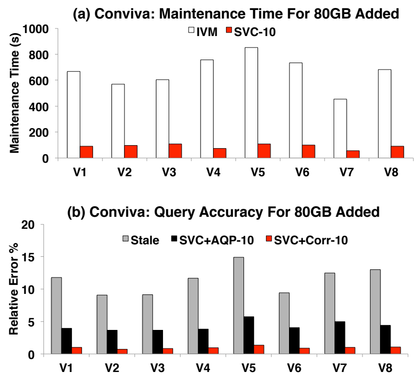


Figure 8: (a) We compare the maintenance time of SVC with a 10% sample and full incremental maintenance, and find that as with TPCD SVC saves significant maintenance time. (b) We also compare SVC+CORR to SVC+AQP and No Maintenance and find it is more accurate.

SVC+AQP and no maintenance.

7.4 Outlier Indexing

In our next experiment, we evaluate our outlier indexing. We note that none of the prior experiments used an outlier index. We index the l_extendedprice attribute in the lineitem table. We evaluate the outlier index on the complex TPCD views. We find that four views: V3, V5, V10, V15, can benefit from this index with our push-up rules. These are four views dependent on l_extendedprice that were also in the set of "Complex" views chosen before.

In our first outlier indexing experiment (Figure 7(a)), we analyze V3. We set an index of 100 records, and applied SVC+CORR and SVC+AQP to views derived from a dataset with a skew parameter $z = \{1, 2, 3, 4\}$. We run the same queries as before, but this time we measure the error at the 75% quartile. We find in the most skewed data SVC with outlier indexing reduces query error by a factor of 2. Next, in $\big($Figure 7 (b)$\big)$, we plot the overhead for outlier indexing for V3 with an index size of 0, 10, 100, and 1000. While there is an overhead, it is still small compared to the gains made by sampling the maintenance strategy.

7.5 Conviva

We derive the views from 800GB of base data and add 80GB of updates. In Figure 8(a), we show that while full maintenance takes nearly 800 seconds for one of the views, a SVC-10% can complete sample cleaning in less than 100s for all of them. On average over all the views, SVC-10% gives a 7.5x speedup.

In Figure 8(b), we show that SVC also gives highly accurate results with an average error of 0.98%. In the following experiments, we will use V2 and V5 as exemplary views. V5 is the most expensive to maintain due to its nested query and V2 is a single level group by aggregate. These results show consistency with our results on the synthetic datasets.

8. RELATED WORK

Sampling has been well studied in the context of query processing [3,13,25]. Both the problems of efficiently sampling relations [25] and processing complex queries [2], have been well studied. In SVC, we look at a new problem, where we efficiently sample from a maintenance strategy, a relational expression that updates a materialized view. We generalize uniform sampling procedures to work in this new context using lineage [12] and hashing. We look the problem of approximate query processing [2,3] from a different perspective by estimating a "correction" rather than estimating query results. Srinivasan and Carey studied a problem related to query correction which they called compensation-based query processing [29] for concurrency control. However, this work did not consider applications when the correction was applied to a sample as in SVC.

In the context of materialized view maintenance, sampling has primarily been studied from the perspective of maintaining samples [27]. Similarly, in [17], Joshi and Jermaine studied indexed materialized views that are amenable to random sampling. While similar in spirit (queries on the view are approximate), the goal of this work was to optimize query processing not address the cost of incremental maintenance. There has been work using sampled views in a limited context of cardinality estimation [19], which is the special case of our framework, namely, the `count` query. Nirkhiwale et al. [24], studied an algebra for estimating confidence intervals in aggregate queries. The objective of this work is not sampling efficiency, as in SVC, but estimation. In a limited setting, where we consider only views constructed from select and project operators, SVC's hash pushdown will yield similar results to their model.

A well studied subject is data cleaning through provenance on materialized views. For example, Wu and Madden [31] studied explanations for outliers in group-by aggregate queries. The technique used to trace the provenance of an outlier row in a result is very similar to the technique proposed in this work. However, this work adddress a different cleaning problem from SVC. In SVC, we look to find a relational expression that update an erroneous row such that it is correct. In [31], the authors remove base data such that the row is no longer an outlier. Also, Challamalla et al. [4] proposed a technique for specifying errors as constraints on a materialized view and proposing changes to the base data such that these constraints can be satisfied. In our setting, we do not know these constraints a priori and we would have to materialize the entire up-to-date view to be able to specify which rows are erroneous. Finally, the work by Meliou et al. [21] also explores tracing lineage through materialized views. Like [31,31], it requires an explicit specification of the errors and also does not suggest a cleaning operation to fix these errors. While SVC has been proposed in the materialized view context, it subsumes the data cleaning work that we previously did in [30]. SVC can be used as an underlying sampling module to extend these data cleaning operations to changing datasets or sampled relations.

Finally, the theory community has has studied related problems. There has been work on the maintenance of approximate histograms, synopses, and sketches [10,14], which closely resemble aggregate materialized views. This work did not model queries on the approximate data structures as in SVC. Furthermore, the goals of this line work (including techniques such as sketching and approximate counting) has been to reduce the required storage, not to reduce the required update time.

9. LIMITATIONS

While our experiments show that SVC works for a variety applications, there are a few limitations which we summarize in this section. There are three primary limitations for SVC: class of queries, types of materialized views, and the specification of maintenance strategy. In this work, we primarily focused on aggregate queries and showed that accuracy decreases as the selectivity of the query increases. Sampled-based methods are fundamentally limited in the way they can support "point lookup" queries that select a single row. This is predicted by our theoretical result that accuracy decreases with $\frac{1}{p}$ where p is the fraction of rows that satisfy the predicate. In terms of more view definitions, SVC does not support views with ordering or "top-k" clauses, as our sampling assumes no ordering on the rows of the MV and it is not clear how sampling commutes with general ordering operations. SVC also requires the maintenance strategy to be parametrized in terms of relational algebra which may not always be possible. Tools like DBToaster achieve some of their performance gains by code generation and not specifying maintenance in SQL.

10. CONCLUSION

Materialized view maintenance is often expensive, and in practice, immediate view maintenance is avoided due to its costs. However, this leads to stale materialized views which have incorrect, missing, and superfluous rows. In this work, we formalize the problem of staleness and view maintenance as a data cleaning problem.

SVC uses a sample-based data cleaning approach to get accurate query results that reflect the most recent data for a greatly reduced computational cost. To achieve this, we significantly extended our prior work in data cleaning, SampleClean [30], for efficient cleaning of stale MVs. This included processing a wider set of aggregate queries, handling missing data errors, and proving for which queries optimality of the estimates hold. Another significant contribution of SVC is our outlier indexing approach which reduces the sensitivity of sampling to skewed data distributions. We presented both empirical and theoretical results showing that our sample data cleaning approach is significantly less expensive than full view maintenance for a large class of materialized views, while still providing accurate aggregate query answers that reflect the most recent data. We evaluate SVC on a real dataset of server logs from Conviva and the TPCD benchmark dataset, and our experiments confirm our theoretical results: (1) cleaning an MV sample is more efficient than full view maintenance, (2) the corrected results are more accurate than using the stale MV, and (3) SVC can be efficiently integrated with deferred maintenance leading to increased query accuracy.

11. REFERENCES

[1] Conviva. `http://www.conviva.com/`.

[2] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. I. Jordan, S. Madden, B. Mozafari, and I. Stoica. Knowing when you're wrong: building fast and reliable approximate query processing systems. In *SIGMOD Conference*, pages 481–492, 2014.

[3] 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 *EuroSys*, pages 29–42, 2013.

[4] A. Chalamalla, I. F. Ilyas, M. Ouzzani, and P. Papotti. Descriptive and prescriptive data cleaning. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 445–456, 2014.

[5] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. R. Narasayya. Overcoming limitations of sampling for aggregation queries. In *ICDE*, pages 534–542, 2001.

[6] S. Chaudhuri and V. Narasayya. TPC-D data generation with skew. `ftp.research.microsoft.com/users/viveknar/tpcdskew`.

[7] R. Chirkova and J. Yang. Materialized views. *Foundations and Trends in Databases*, 4(4):295–405, 2012.

[8] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009.

[9] L. S. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey. Algorithms for deferred view maintenance. In *SIGMOD Conference*, pages 469–480, 1996.

[10] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.

[11] D. R. Cox and D. V. Hinkley. *Theoretical statistics*. CRC Press, 1979.

[12] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *VLDB J.*, 12(1):41–58, 2003.

[13] M. N. Garofalakis and P. B. Gibbons. Approximate query processing: Taming the terabytes. In *VLDB*, 2001.

[14] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. *ACM Trans. Database Syst.*, 27(3):261–298, 2002.

[15] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.

[16] H. Gupta and I. S. Mumick. Incremental maintenance of aggregate and outerjoin expressions. *Information Systems*, 31(6):435–464, 2006.

[17] S. Joshi and C. M. Jermaine. Materialized sample views for database approximation. *IEEE Trans. Knowl. Data Eng.*, 20(3):337–351, 2008.

[18] C. Koch, Y. Ahmad, O. Kennedy, M. Nikolic, A. Nötzli, D. Lupei, and A. Shaikhha. Dbtoaster: higher-order delta processing for dynamic, frequently fresh views. *VLDB J.*, 23(2):253–278, 2014.

[19] P.-A. Larson, W. Lehner, J. Zhou, and P. Zabback. Cardinality estimation using sample views with quality assurance. In *SIGMOD*, pages 175–186, 2007.

[20] P.-Å. Larson and H. Z. Yang. Computing queries from derived relations. In *VLDB*, pages 259–269, 1985.

[21] A. Meliou, W. Gatterbauer, S. Nath, and D. Suciu. Tracing data errors with view-conditioned causality. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, pages 505–516, 2011.

[22] M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2):226–251, 2003.

[23] M. Nikolic, M. Elseidy, and C. Koch. Linview: incremental view maintenance for complex analytical queries. In *SIGMOD Conference*, pages 253–264, 2014.

[24] S. Nirkhiwale, A. Dobra, and C. M. Jermaine. A sampling algebra for aggregate estimation. *PVLDB*, 6(14):1798–1809, 2013.

[25] F. Olken. *Random sampling from databases*. PhD thesis, University of California, 1993.

[26] F. Olken and D. Rotem. Simple random sampling from relational databases. In *VLDB*, pages 160–169, 1986.

[27] F. Olken and D. Rotem. Maintenance of materialized views of sampling queries. In *ICDE*, pages 632–641, 1992.

[28] E. Rahm and H. H. Do. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.

[29] V. Srinivasan and M. J. Carey. Compensation-based on-line query processing. In *SIGMOD Conference*, pages 331–340, 1992.

[30] J. Wang, S. Krishnan, M. J. Franklin, K. Goldberg, T. Kraska, and T. Milo. A sample-and-clean framework for fast and accurate query processing on dirty data. In *SIGMOD Conference*, pages 469–480, 2014.

[31] E. Wu and S. Madden. Scorpion: Explaining away outliers in aggregate queries. *PVLDB*, 6(8):553–564, 2013.

[32] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo. The analytical bootstrap: a new method for fast error estimation in approximate query processing. In *SIGMOD*, pages 277–288, 2014.

[33] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. In *SIGMOD Conference*, pages 265–276, 2014.

[34] J. Zhou, P.-Å. Larson, and H. G. Elmongui. Lazy maintenance of materialized views. In *VLDB*, pages 231–242, 2007.