Efficient and Reusable Lazy Sampling

Viktor Sanca
EPFL
viktor.sanca@epfl.ch

Periklis Chrysogelos[*]
Oracle
periklis.chrysogelos@oracle.com

Anastasia Ailamaki
EPFL
anastasia.ailamaki@epfl.ch

ABSTRACT

Modern analytical engines rely on Approximate Query Processing (AQP) to provide faster response times than the hardware allows for exact query answering. However, existing AQP methods impose steep performance penalties as workload unpredictability increases. While offline AQP relies on predictable workloads to a priori create samples that match the queries, as soon as workload predictability diminishes, returning to existing online AQP methods that create query-specific samples with little reuse across queries results in significantly smaller gains in response times. As a result, existing approaches cannot fully exploit the benefits of sampling under increased unpredictability.

We propose LAQy, a framework for building, expanding, and merging samples to adapt to the changes in workload predicates. We propose lazy sampling to overcome the unpredictability issues that cause fast-but-specialized samples to be query-specific and design it for a scale-up analytical engine to show the adaptivity and practicality of our framework in a modern system. LAQy speeds up online sampling processing as a function of data access and computation reuse, making sampler placement after expensive operators more practical.

1 Reducing the Data Volume with Sampling

Data exploration is crucial to deriving insights and informed decisions in today's data-driven world. Visualization tools and interactive dashboards provide a convenient and rich exploration interface. Nevertheless, humans require fast responses to maintain their focus during mental tasks: Miller [22] reports a 0.1 seconds response window for users to feel the UI follows their actions and 15 seconds as a hard limit to avoid demoralization and breaking the line of thought. However, with current memory technologies providing a few hundred GBps of memory bandwidth, analytical engines cannot even process simple queries that touch more than a few GBs of data in the 0.1 seconds window, despite running on TB-sized memories.

Analytical engines have long relied on approximate query processing to reduce the data processing time [2, 19, 6, 3, 27, 24, 8, 5, 29, 15]. Offline approximate query processing methods prebuild samples to reduce the data access and processing time during query execution [2]. However, the significant savings of such approaches come at the expense of requiring predictable workloads. Such predictable workloads appear in data warehousing operations, but they mismatch interactive workloads like data exploration, where queries constantly change [18]. Online query processing generalizes offline sampling into interactive use cases. Instead of relying on query templates, it does the sampling during query execution before expensive operations. Such approaches reduce the processing time; however, sampling during query execution is a heavy operation [29, 3]. To minimize this cost, existing approaches rely on pushing lightweight, selective operations below sampling [15] and sample caching [24]. However, specializing the online sample to the current query reduces its potential to be suitable for another query. As a result, existing approaches introduce a steep trade-off between sample reuse and sampling overhead for interactive exploration.

This work introduces LAQy, a sampling-based AQP framework that increases the sample reuse opportunities while maintaining the sample creation to the same or lower cost than online sample creation. LAQy relaxes the sample matching requirements by allowing samples to match a query partially. This relaxation allows using samples that would otherwise be considered inappropriate for an incoming query – resulting in significant execution time savings. LAQy corrects the query-sample mismatch using *delta* samples: samples that LAQy uses to augment the partially matching sample with the missing pieces needed to satisfy the query approximation requirements. As a result, LAQy extends i) the effectiveness of online sampling techniques to a greater range of query workloads, ii) provides a system-level acceleration technique that maintains the theoretical sample properties, and iii) reduces the query predictability requirements that existing systems need to overcome the hardware limitations through query approximation techniques.

Overall, LAQy improves the efficiency of online approximate query processing systems by increasing sample reuse and bridging the gap between predictable and unpredictable predicates in approximate query processing. As a result,

[*]Work done while the author was at EPFL

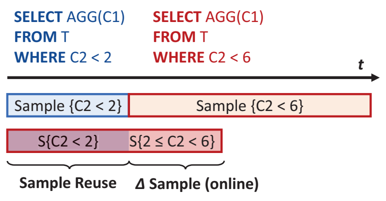


Figure 1: Relaxing the predicate predictability allows reuse.

LAQy enables fast responses despite the query unpredictability that characterizes data exploration workloads, which previously hindered the effectiveness of query approximation methods. While most prior related systems use sampling in disk-based, single-threaded, and distributed setups, we investigate and address the bottleneck shift in scale-up single-machine setups with minimal engine modifications. LAQy proposes a lazy sampling algorithm to avoid costly sample creation through an in-memory-friendly architecture and judicious workload-driven sample construction and merging. This allows for speeding up the sampling operation for base relations and also enables reuse opportunities when samplers must be placed after joins, for example, when meaningful filtering and sampling dimensions are only available after joining the fact tables with dimension tables.

2 Online Flexibility and Offline Predictability

Exploratory data analysis creates hard-to-predict query patterns, yet its interactive nature requires fast response times to keep the data analyst focused and productive. Further, keeping the user engaged requires response times exceeding the underlying hardware's capabilities. As a result, analytical engines have relied on approximate query processing to reduce the data processing cost. A core requirement of AQP methods is that when the query implies some grouping, all groups are represented in the output. To achieve that, variants of approximation methods often rely on stratified sampling: the systems analyze the query clauses and extract the columns that, if not included in the stratification key, the query output could sample out tuples. These columns are called the *Query Column Set* (QCS), and when aligned with the query requirements, they allow for strict bounds on the error of the computed aggregations [2]. Furthermore, Quickr [15] provides optimization rules for injecting the samplers in the query plan and transforming samples and their QCS requirements while pushing them across various relational operators. Conversely, the remaining non-QCS columns are called *Query Value Set* (QVS).

Workload predictability. The applicability of the different approximate query processing methods depends on workload predictability. Agarwal et al. [2] classify workloads into four categories based on the predictability of the queries: i) *predictable queries* where the upcoming queries are known, e.g., monthly or weekly warehousing tasks that repeat the same query in a fixed interval, ii) *predictable query predicates* where the *filter conditions* of upcoming queries are fixed and known, iii) *predictable QCSs* where the *grouping* or *filtering* columns are known, but the actual filtering values are revealed only during the query invocation, and iv) *unpredictable queries* where there is no information about the upcoming queries.

While the query patterns during data exploration are hard

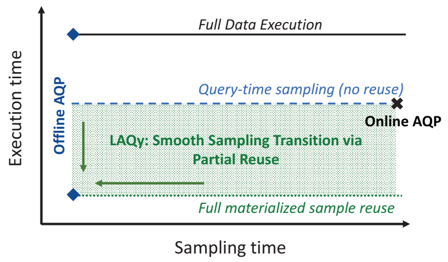

Full Data Execution
Query-time sampling (no reuse)
Online AQP
LAQy: Smooth Sampling Transition via Partial Reuse
Full materialized sample reuse
Offline AQP
Execution time
Sampling time

Figure 2: The design space of sampling-based AQP methods.

to predict, they are also not entirely unpredictable. Users often add, remove, expand, or shrink filters, grouping columns and joined tables to focus their exploration on specific subsets of the input or expand to increase their exploration scope and hypothesis testing [30]. As a result, the analytical engine often receives incremental changes to the query shape, placing such workloads between predictable, or slowly changing, QCSs and predictable query predicates. Yet, approximation techniques for predictable query predicates are much more efficient as samples are handled similarly to offline views and thus incur small overheads. In contrast, (mostly) predictable QCSs rely on online sampling.

Issue #1: strict sample matching rules. Whether a sample subsumes a query or not is, currently, a binary classification: either a query has an appropriate QCS and QVS, or it can not provide the necessary guarantee, even if just one column is not contained in one of those sets. As a result, small mismatches in column set requirements create a new sample – with the corresponding overhead and missed opportunities.
Example. Figure 1 shows two queries arriving in a sequence, with the second having an expanded filter, similar to the queries submitted by a data scientist who zooms out to cover a greater input range. Suppose that, during query execution, a sample on T is created after the filtering condition, e.g., to minimize the sampling cost. That sample will not have the information necessary to answer the expanded (red) query. As a result, while the two queries are very similar, the sample would be rejected, and a new one would be created, reducing the first sample's effectiveness.
Challenge #1: increasing the sample usefulness. Efficient query approximation techniques are needed to increase the usefulness of each sample by relaxing the sample matching requirements despite the theoretical requirements. To avoid compromising the theoretical sample properties, LAQy increases the sample usefulness by allowing samples to partially satisfy a query and creating delta queries that require smaller samples for their approximation.

Issue #2: creating a new sample is costly. When existing samples do not match the current query, a new sample is created, or approximate processing is abandoned altogether. To mitigate such overheads, existing methods [24] aggressively cache samples and synopsis. Specifically, Taster [24] continuously inspects the workload to materialize samples as a side-effect of execution for future (re)use based on the recent query history. However, the materialized sample is only helpful if it entirely subsumes the predicates and QCS, otherwise falling back to the online AQP processing. The new sample is built from scratch despite potentially having similar samples. As a result, despite an existing sample satisfying part of the query, the new sample will be built on the entire input and have the full QCS size, which prior work has shown can have a prohibitive cost, especially as the QCS and QVS sizes increase [29].

Example. If the queries of Figure 1 were also grouped on C2, then the corresponding samples would have C2 in the QCS. Although the blue sample is a subset of the sample required for the red query, existing approaches would discard the blue sample when evaluating the second (red) query. Further, with the valid range of C2 increasing, building the second (red) sample with stratification on C2 would be significantly slower than making the blue sample, as the number of strata is the dominating factor during sample creation [29].

Challenge #2: efficiently reacting to missing samples. Efficient query approximation techniques are needed to build the required samples quickly and efficiently. To reduce the sample creation time, LAQy judiciously builds only the missing parts of a required sample and retrieves the rest from the already available samples.

Issue #3: unpredictability increases the risk of useless pre-sampling. Offline sampling [2] mitigates the sampling overhead by sampling before the query time. During query processing, the sample is already available, effectively eradicating the cost of sampling from the response time. Further, to avoid the overhead of periodically recreating the sample as the source data are updated, Birler et al. [3] propose incrementally maintaining the sample during updates. Yet, offline methods require that the useful samples are known before query time. To reduce the probability of wasted samples, offline approaches tend to build more inclusive and, thus, bigger samples. However, this is still a double-edged sword. While it may support more queries, 1) using a bigger sample incurs a higher cost for each query using it, 2) a higher build cost, and yet the potential of mispredicting the query workload and not using the sample.

Example. Consider this time that an aggressive sampling approach sees the condition on C2, and instead of applying the filter before the sampling, it decides to stratify on C2 as well (include C2 in the QCS). While this would make the sample reusable in the second (red) query, it may not pay off if, for example, multiplying the number of strata by the cardinality of C2 makes sample creation much slower than creating two different samples – a common occurrence for high cardinality columns [29].

Challenge #3: sampling without regret. As a result, the overgeneralization of samples may result in significant costs. Efficient query approximation systems need to minimize the risk of wasteful sample creation by reducing the probability of creating a minimally useful sample. To overcome this issue, LAQy builds only samples that will be immediately used to accelerate a query.

Summary. The current rigidness of whether a sample satisfies a query, combined with the high cost of sample creation and the uncertainty about the long-term gains of creating bigger samples, highly affect the effectiveness of online approximation techniques. Further, data exploration often results in partially overlapping queries and small transitions across predicates, resulting in the aforementioned issues for such workloads. We describe how LAQy's design allows tackling all three issues through a design that focuses on maximizing the (even partial) sample reuse to minimize the sample creation (through lazy delta samples) and a sample-as-you-query model that reduces the regret for building a sample by creating only samples that are immediately useful. As a result, LAQy bridges the gap between online and offline approximation methods by partial sample reuse (Figure 2).

3 The Design Principles of Lazy Sampling

LAQy is an approximate query processing engine designed to bridge the gap between offline and online AQP methods in scale-up systems. Existing approaches have to select between building low-overhead samples specialized to the query predicates or paying a higher sample construction overhead to create reusable samples. Instead, LAQy achieves a sweet spot between reusability and sampling time by taking advantage of the mergeable nature of samples. Furthermore, LAQy is compatible with adaptive storage and sample budgeting solutions, like Taster [24], to allow adaptive management of the allocated sample space based on workload patterns.

Instead of imposing a binary can-or-cannot-be reused per-sample decision, the samples generated by LAQy create a continuous spectrum between online and offline sampling methods (Figure 2). As a result, samples do not have to be entirely ignored; instead, they provide a partial input to the query. Furthermore, for the query input that is not covered by the existing sample, LAQy avoids building a full sample. Instead, it constructs only the part of the sample necessary for the current query – minimizing the overhead imposed by sample construction, resulting in better applicability of our method in the era of in-memory engines that saturate the available memory bandwidth.

The three core principles of LAQy enable it to directly address the challenges outlined in Section 2:

Principle #1: Partial reuse. Predicates are traditionally considered either known or too volatile to specialize the sample to the predicate value – creating a steep penalty, even if the predicates are not entirely random. For example, typical data exploration patterns [30] imply that the focus of interest may change but correlate to the initial scope of analysis, albeit in unpredictable ways. LAQy exploits the overlap across the predicates to reuse the existing materialized samples and compute only the delta samples to satisfy uncovered, non-overlapping sample ranges, extending the strategy of offline sampling systems.

Principle #2: Judicious sampling. Stratified sampling is an expensive operation. Stratification imposes random accesses making sampling potentially as expensive as joins or aggregations. Nonetheless, it reduces the input to upcoming operations and the execution time of future queries if the sample is materialized and reused. To avoid excessive overheads without a corresponding long-term speedup, LAQy minimizes the input of stratification operations to the minimum required for the current query, along the direction of the online sampling systems.

Principle #3: Laziness and minimal waste. Every bit counts, but some bits count more. The long-term gain of a sample depends on upcoming queries. While the prediction accuracy of future queries can vary, samples created for the current query have an immediate turnaround. Also, we can give a higher turnaround to already created samples by increasing their utility through partial reuse. LAQy reduces the

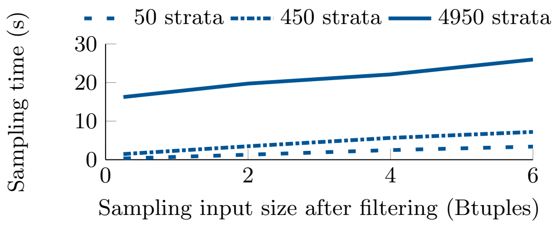


Figure 3: Impact of #tuples of the dataset and #strata defined by the QCS on the stratified sample creation time.

decision-making and pushes it as late as possible by building only the necessary delta samples and merging them only when needed. This makes our approach complementary to both the online and offline state-of-the-art sampling-based AQP systems.

4 Pruning the Sample Construction Space

LAQy bridges the gap between offline and online sampling by exploiting the overlap of query predicates to reuse and merge samples. Section 4.1 starts by evaluating the different parameters that affect the build time for a stratified sample. Section 4.2 links the various parameters with the query predictability. Lastly, in Section 4.3, it motivates our main observation: relaxing the predictability requirements for predicates allows lazily building samples and amortizes the (stratified) sample build time across queries with overlapping predicates.

4.1 Building Stratified Samples

To create a stratified sample, each input element is inserted into a reservoir based on the element values on the QCS columns. Each stratum thus keeps track of the reservoir and the number of considered elements (*weight*) as the admission probability depends on the number of previously considered items.

Considering an input item for inclusion into a reservoir generally requires random access to find and update the *admission-control* state (*random number generator*). If the item is admitted, then finding and replacing an existing item from the reservoir requires one more random access. The latter may or may not be on the same cache line as the admission state, depending on the reservoir capacity (k) and whether the reservoir's storage is inlined with the admission state.

Admission control occurs for every tuple, so the corresponding time is reduced as the input tuples decrease. Admission, however, happens stochastically with a probability that converges to the sampling rate as more items are considered per reservoir. As a result, admissions are responsible for a small portion of the build time. Furthermore, following the observation that an item will be infrequently replaced, LAQy does not pack the reservoir storage with the admission state. Instead, it stores a pointer to the actual reservoir together with the admission state to reduce the footprint of the hash table data structure used to store the strata [29].

Figure 3 shows how the build time increases with an increase in the input size. Independently of the number of strata, the sampling time follows a similar trend. However, as the number of strata grows, we observe a higher sampling time, even for small input sizes. Each stratum introduces a

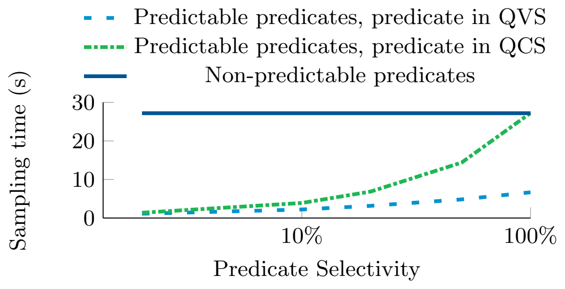


Figure 4: Stratified sampling time for various selectivities.

constant allocation and initialization even if it does not reach capacity. As a result, while the input size affects the sample building time, strata initialization time, count, and random accesses to the stratum state significantly impact the sample build time, exacerbated by the total cardinality of QCS.

Key observations & predicate predictability. LAQy builds on top of two observations. First, the number of strata and tuples has the highest impact on sampling time, while per-stratum capacity k has a lower impact on the build time. Second, increases in the reservoir capacity have a marginal impact on the overall sampling time while controlling the desired error bounds by obtaining sufficient sample support. LAQy takes advantage of these two observations to accelerate sampling for unpredictable predicates by exploiting patterns and overlaps present in the query predications that reduce the penalty of mispredictions in the case of partial predicate matches.

4.2 The Cost of Predicate Unpredictability

Suppose we have the following query that we would like to approximate by creating a sample, where the input relation T can be a base table or a subquery result.

```
SELECT C1, SUM(C2) FROM T
WHERE C3 > 5 [AND C1 IN ('G1', 'G2')]
GROUP BY C1
```

If the predicate values are predictable on C3 and C1, the future queries are assumed to satisfy those conditions. This means that building a stratified sample on QCS={C1} with filters pushed down before the sampler can answer the future queries with equal or *stricter*, subsuming predicates[1].

If the predicate values are not predictable, but predicate columns are, those columns are added to the QCS without pushing down the filters before sampling. This makes the resulting sample applicable to any predicate on those columns at the cost of a larger input and more complex stratified sampling with multiple columns in QCS. As the cost of sampling is not negligible at query execution time, the cost of more complex sampling schemes adds a prohibitive penalty. The user may be interested only in limited, yet unpredictable, dataset regions, incurring wasteful precomputation at a critical execution path. Suppose this happens if the user is interested only in specific groups in the future queries with a filter on C1 in brackets.

To demonstrate the impact of selecting one of the above options, Figure 4 shows the sample build time required for

[1]If a sufficient sample size support exists for requested error guarantees after applying the stricter predicate [10, 2, 15].

creating a stratified sample for different selectivities. Section 6 provides the detailed input and hardware details. We start with a stratified sample on QCS={`C1`} and filter pushdown on `C3` (*predictable predicates on QVS*), which results in 450 strata. To improve the reusability of the materialized sample without any predictions about the runtime predicate values, the column `C3` is added to QCS without any filter pushdown, resulting in 4950 strata (*non-predictable predicates*).

However, when the predicate is on a column `C1` that would participate in the QCS due to being part of the `GROUP BY` clause, the filter can be pushed down along with adding the column to the QCS. In the best case, on the highly selective end for the newly added QCS, this approach prunes the sample input instead of making a strict decision to add the entire column to the QCS (*predictable predicate on QCS*). There is up to 19-24x slowdown to create a non-predicate-specific sample, representing an average slowdown of 6.7-11x across the selectivities to resolve the predicate value unpredictability with the **existing all-or-none sample matching** systems and approaches. The goal is to construct a reusable sample as the highest speedup is achieved using a previously materialized sample (offline AQP) instead of online sampling. The leading cause of the predicate predictability rigidity is the strict requirement of predicate subsumption for reuse. **The sample reuse dichotomy** creates a clear-cut tradeoff between reusability and performance, driven by predictions instead of the actual workload with *overlapping* predicates.

4.3 Relaxing Sample Predicate Predictability

There is, however, a middle ground between predictable and unpredictable predicates. Consider the two queries in Figure 1 to demonstrate this. If the value for predicate on `C2` is known ahead of time, the query belongs in the predictable predicate category. If that value varies, then the predicate would be considered unpredictable. Suppose a sample is materialized as a side-effect of workload. In that case, the sample can be reused if the predicates are subsumed [24], assuming enough tuples in the sample satisfy the predicate to achieve the desired error guarantees.

To our best knowledge, existing systems and algorithms opt to rebuild samples if the predicate is not fully subsumed, even if generating an online sample for the missing range $[2, 6)$ would suffice to answer the query using the previously materialized, *effectively offline* sample, as in Figure 1.

We call a sample generated for the uncovered range $(x, y]$ a Δ *(delta)* sample, the process of generating such sample *expansion* as it expands the samples' predicate coverage, and the process of combining the reusable sample with Δ sample *merging*. We call the overall sampling *lazy*, as it defers and relaxes the strict predictability rules as late as possible while reducing the work that samplers need to do.

Summary. Our approach relaxes the predictability requirements to improve the reuse of previously materialized samples through workload-aware methods described by prior work [2, 24]. We modify the physical sampling algorithm and obtain an equivalent *logical* sample with the requested characteristics. We explore the reservoir-based sampling algorithms (simple reservoir sample, stratified reservoir sampling) commonly used in systems, but mainly since *merging* two or multiple reservoirs preserve the characteristics of the final (reservoir) sample [33]. We propose a method that bridges *offline* and *online* sampling, **complementary** to the prior art.

5 Lazy Sampling with LAQy

Traditionally [2, 15], the columns in a stratified sample are split into two sets: the QCS (*Query Column Set*) and the QVS (*Query Value Set*). The QCS contains the columns that affect the participation of the rows in the output, such as columns participating in filters, join conditions, and grouping columns. The QVS has the remaining columns, such as the aggregation columns.

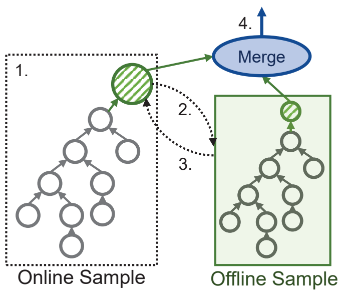


Figure 5: LAQy: the relaxed sample reuse framework and steps.

We describe the main steps of how LAQy relaxes the predicate requirement for sample reuse, depicted in Figure 5:

1. Starting from an approximable query, an optimizer (e.g. [15, 24]) decides on the *logical* sampler placement (striped green circle). The sampler can have as input base relations or subqueries in a more general case.
2. The sample store attempts to find a materialized sample with the requested characteristics of the *logical* sampler.
   (a) If there is a sample that already covers the predicate space with a subsuming predicate, we can use the existing sample if sufficient sample support exists after filtering, fully reusing the *offline sample*.
   (b) If no sample exists overlapping in predicates and requirements, *online sampling* is performed only.
   (c) If a partially overlapping sample exists, calculate the non-overlapping predicate for the Δ query and prepare the sample for merging in the query execution pipeline. Proceed to the next step (3).
3. The predicates are pushed down the original query plan to initiate online Δ sampling.
4. Online and offline (stratified) reservoirs are merged, producing the equivalent logical sample.

LAQy combines samples and triggers the generation of a new partial (delta) sample only for the relevant input data not covered by the existing samples. Thus, for each sample, LAQy maintains the *Query Predicate*, *Query Input*, *QCS* and *QVS* that represent the (*logical*) input of each sampler (the striped circle, Offline Sample in Figure 5). Making this information part of the sample description makes explicit that the sample is malleable and reusable based on specific conditions outlined in the rest of this section.

To combine two samples, LAQy relies on selecting samples that cover the area of interest but do not overlap. Consider a query predicated on $C3 > 2$. It can be served by combining a sample on $C3 > 5$ with a sample on $C3 \in (2, 5]$. However, combining a sample on $C3 > 5$ with a sample on $C3 \in (2, 7]$ would sample tuples in the $C3 \in (5, 7]$ range twice, introducing a bias towards those intervals – violating the per-reservoir uniformity assumption. To resolve this, LAQy *extends* samples by pushing down the predicates to merge reservoirs on non-overlapping predicates in QVS, reducing the sampler input as a desirable property of Δ samples.

LAQy relies on prior art that analyzes and proposes solutions for optimization rules for sampling for online AQP [15], as well as workload-based predictions and sample materializations [2, 24]. We aim to bridge and relax the dichotomy between the two approaches via flexible sample reuse. Therefore, we focus on a solution that synergetically works with the prior art and present and evaluate our contributions within reasonable starting assumptions. At either extreme, our system would run as purely *online* or *offline* sampling-based AQP system. For example, we will assume that an offline sample exists to focus on the behavior and performance of the proposed relaxed reuse via lazy sampling.

5.1 Sample Merging

Reservoir sampling is amenable to updates and maintenance while preserving the requested sample characteristics. We use the property of well-defined *merge* behavior to enable both the independent, data-parallel scale-up execution as well as the key contribution that relies on sample merging. Different data processing operations, such as filtering or stratification, impact the data distribution. Further, partitioning and/or splitting the data for parallelization potentially introduces additional data skew. As a result, using samples created after such operations requires careful consideration to avoid creating biased samples.

LAQy uses the weighted reservoir sampling algorithm [4] to recover a sample equivalent to a full resample of the input. Specifically, during merging, LAQy weighs the elements of the to-be-merged samples based on what proportion of the input they represent. To do so, each created sample keeps the reservoir R and tracks the running sum of (importance) weights w of previously qualifying elements. This allows LAQy to use the exact reservoir weights as it creates and stores samples just in time, effectively maintaining the count of the elements as the weight of each qualifying input element is one. Using these (importance) weights, LAQy calculates the new weights associated with every sample to use them further with the weighted reservoir sampling algorithm so that the elements of the merged sample have the same weight as after a full resample.

The key observation is that two independent reservoirs with their associated weights $\{R_1, w_1\}$ and $\{R_2, w_2\}$ can be *merged* to obtain $\{R_m, w_1 + w_2\}$. Intuitively, reservoir R_1 represents w_1 tuples, R_2 represents w_2 tuples and a reservoir R_m equivalent to sampling the union of original input data of R_1 and R_2 would have combined weights $w_1 + w_2$. To avoid resampling the original input data, we use the existing samples where we adjust the sampling weight from uniform to biased, as in general case weights are different, where elements of R_1 are selected with probability $\frac{w_1}{w_1+w_2}$; where converse holds for elements of R_2 [33]. More formally, to combine reservoirs R_1 and R_2 into R_m, we perform weighted reservoir sampling [4] with R_1 and R_2 and their weights as input to the algorithm. The result is equivalent to performing reservoir sampling over the combined input while avoiding repeating processing and accessing the original data represented by R_1 and R_2. Therefore, merging itself does not change the properties of the obtained reservoir as it is a reformulation of the inputs to the algorithm that is amenable to changing the input weights. Query-driven sampling and operator placement, even past expensive operations, results in samples where the error estimation framework is compatible with principles established by prior art [8, 15, 10].

Since, in general, even the reservoir sizes k may differ, we introduce *Scaled Proportional Sampling* to further bias the weight by the ratio of $k1$ and $k2$, obtaining the weight factor of $\frac{k_{scaled}}{w}$ to achieve proportional reservoir merge via weighted reservoir sampling of the inputs. LAQy maintains reservoirs with their weights, and the reservoir sizes are parameters known at runtime.

5.2 Issuing Δ Samples with Relaxed Predicates

LAQy differentiates between two general types of reuse across the samples: tightening and relaxing the predicates depending on the available materialized samples that may require issuing a Δ sampling query.

Conditional transition to stricter predicates happens when a sample already covers the wanted space, but the predicate of interest is stricter. We can use the existing sample under the condition that samples have enough support after the predicate pushdown to satisfy the accuracy requirements [24]. No Δ sample needs to be issued if the sample meets the condition; otherwise, a complete online sampling must be performed to satisfy the guarantees.

Relaxing predicates requires adding input tuples for predicate values not covered by an existing sample. We issue a Δ query based on the predicates to find the *inverted*, non-overlapping interval.

Combined tightening and relaxing is the case when predicates require tightening of the predicates on an existing *offline* sample and relaxing via executing the Δ sample for the missing interval. However, the sample support (of each stratum) after applying the predicate $\sigma_{predicate}(S)$ is unknown ahead of time, which may impact the (specified) expected error bounds. If strict qualifying sample support is needed, we can follow the conservative approach: an online query is subsequently executed for all the reservoirs/strata that do not have sufficient support, which performs sampling after the filter pushdown. This validates if the *exact* low (or lack of) support comes from the original data distribution or as the artifact of sampling. We can continue the query in a less strict setting and report the obtained error bound with the available data. One approach to reducing the probability of insufficient sample support is introducing an oversampling factor $\alpha \geq 1$ to create reservoirs sized $\alpha \cdot k$. This trades-off space for tentatively higher sample reusability in case of strict predicates, similar to how past approaches [2] create samples with different parameters k.

Predicate relaxing allows for building only a Δ sample on the missing value range. LAQy reduces the input to the expensive sampling operations by minimizing the necessary work at query time. We reduce the wasteful processing while taking a sampling decision as late as possible to process only the data of interest to the user, maximizing the reuse of previously materialized *online* or *offline* sampling effort.

6 Evaluating LAQy in a Scale-Up Engine

We implement LAQy in Proteus [28, 16, 7], a parallel in-memory DBMS engine. Proteus uses LLVM to generate customized code for each query, as in the prior work on JIT engines [23, 7, 29, 21]. As Proteus did not have AQP support, we extended it by 1. introducing sampler operators with the corresponding code generation routines, 2. adding a sample lifetime management module that captures the generated samples to allow reuse on subsequent queries, and 3. implementing sample merging aggregation functions and operators designed for scale-up execution.

Hardware setup. We run our experiments on a server with dual-socket Intel(R) Xeon(R) Gold 5118 CPU (2x12 cores, 2x24 threads, HyperThreading enabled) with 384GB DDR4 RAM. All the experiments run on 48 threads.

Dataset. We use the Star Schema Benchmark [25] data for our experiments, using the scale factor (SF) 1000 in a binary columnar layout. We add a unique identifier column (`lo_intkey`) to the `lineorder` table as an 8-byte integer value ranging from 0 to the number of elements in the table, randomly shuffled to enable fine-grained selectivity control without implying a specific data ordering. The necessary columns are preloaded in memory for the experiments.

In line with findings from Microsoft's production big-data cluster [14], where authors report that 90% of the input column sets have between 1 and 6 columns, for our evaluation, we use 1 to 3 columns to produce up to 4950 strata.

Workload. We use two representative query templates to evaluate the holistic performance of online sampling in LAQy. Equation (Q1) describes a scan-heavy query where the sampler is pushed down to the scan operator. Equation (Q2) describes a query with a random-access pattern due to joins with dimension tables that enable sampling and stratifying on other meaningful dimensions, where the sampler is placed higher in the query plan. For both queries, we control and report the selectivity over the fact table (`lo_orderdate`) via the `lo_intkey` attribute.

```
SELECT AGG() FROM lineorder
WHERE lo_intkey BETWEEN(lower AND upper)   (Q1)
GROUP BY lo_orderdate
```

We generate two query sequences to simulate an exploratory workload with changing predicates. The first sequence represents a `long-running` analysis: the user is running the specified query template over 50 iterations such that they progressively extend the value range and narrow it down or use the same interval at a specified rate r. The second sequence represents when the user changes the focus of interest during their analysis, where 60 queries are split into 3x20 batches that we name `short-running` analyses, which are similar in setup to the `long-running` sequence.

```
SELECT AGG() FROM lineorder, date, supplier, part
WHERE lo_intkey BETWEEN(lower AND upper)
AND s_region="AMERICA" AND p_category="MFGR#12"
AND ...  (JOIN) GROUP BY (d_year,p_brand1)
                                              (Q2)
```

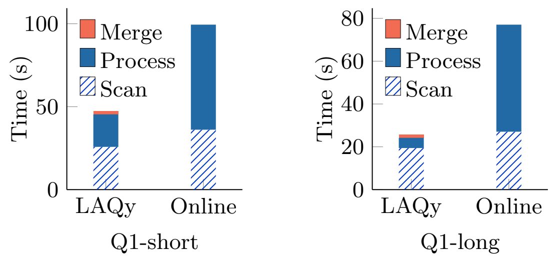


Figure 6: Cumulative processing time breakdown.

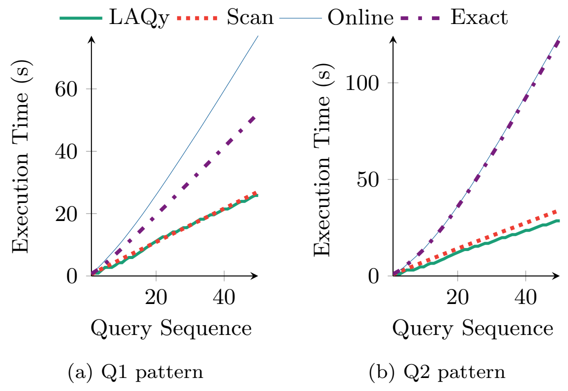


Figure 7: Long query sequence, cumulative execution time.

6.1 Reducing the Cost of Online Sampling

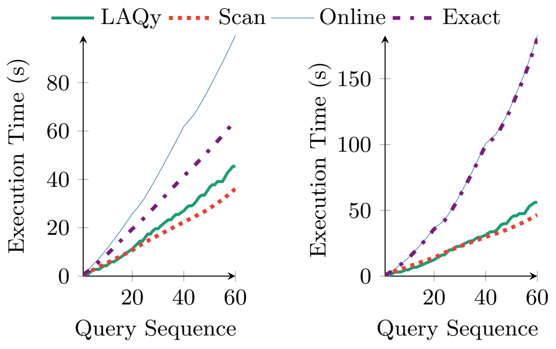
LAQy benefits from reuse opportunities due to overlapping predicates, which enable lazy sampling speedup. In concrete terms, we first present the breakdown of the cumulative processing time of Q1 in Figure 6. First, scan time is lower in the case of LAQy due to detecting full sample reuse opportunities. Second, the processing time (excluding scan) is lower due to lazy sampling and issuing only required Δ samples. Finally, there is a negligible merge overhead to produce an equivalent sample since merging the reservoirs operates over the data samples.

6.2 Efficient & Reusable Sampling with LAQy

In case there is a good workload prediction, offline sampling methods have the potential to create a reusable set of samples. In case the workload is unpredictable or evolving, our approach reduces the misprediction penalty by creating additional samples only over the queries and data relevant to the user - which is definitely known at runtime. We conclude the experimental analysis of LAQy by observing the big picture of cumulative execution time.

6.2.1 Long-running sequence: high reuse

Lazy sampling can achieve the cost effectively below the scan time for samplers pushed down to the base relations (Figure 7a). This is due to combining offline sample use that does not require a scan and progressively creating missing sample components. Equally, this could be the case of a good workload prediction where slight predicate deviations occur that our approach can gracefully fix. In a scan-heavy query, regular online sampling is slower than the exact counterpart, as it introduces sample processing overhead on top of the input coming in at a memory-bandwidth rate (scan).



(a) Q1 pattern (b) Q2 pattern

Figure 8: Short query sequence, cumulative execution time.

Similarly, when the sampler is placed further up in the query plan as in Figure 7b, the combined savings from full offline reuse and selectivity-driven partial reuse enable expensive online sampling costs to be reduced to a scan. Online sampling cost is the same as the exact execution since the input to both operators has a lower throughput due to preceding joins, where additional online sampling processing is masked with the input data rate.

6.2.2 *Short-running sequence: adapting to change*

With moderate reuse opportunity, LAQy enables lazy sampling between the scan and the input rate defined by the preceding operator(s). In particular, when the sampler is fully pushed down, the lazy sampling enables cumulative execution time comparable to sequential scan (Figure 8a). Otherwise, the cumulative cost of lazy sampling is between the scan and the input rate of input operators, reducing both the impact of random data access patterns of preceding operations and the sampling cost through delta sampling and predicate pushdown synergetically, as we show in Figure 8b.

Takeaway. LAQy enables sampling methods to convert the dichotomy of reuse based on full match into a flexible spectrum based on partial matching. As a result, LAQy expands the applicability of existing sampling schemes to a greater query set with a minimal processing overhead.

7 Related Work

Approximate query processing has been of interest over the years as a method to trade-off accuracy for reduced execution times and has found applications in interactive data exploration [18], data visualization [17], cardinality estimation for query optimization [11], and in novel execution schemes such as speculative execution [31, 13].

The prior work and research in the field of AQP is vast and covers the theory, systems, operators, and approximation schemes [8], well summarized in a survey [19]. We limit the scope of our paper to sampling-based approximations in the context of modern scale-up analytical systems, and we briefly present the related work.

Offline AQP systems [26, 2, 12, 20] build samples and data summaries ahead of time, based on assumptions of knowing the future workload and static or slowly changing data. However, when an adequate sample is unavailable, fallback to regular execution or online sampling is often necessary, reducing the expected speedups. LAQy uses the previously created samples to avoid extensive sample creation and to reduce the size of new samples. As a result, LAQy's continuous sample creation increases the potential of having the required sample ready and thus brings online approximation methods closer to the benefits of offline ones.

Online AQP systems [15, 14] perform sampling at runtime and offer speedup through data reduction by injecting samplers at data-intensive parts of the query plan. As the original data needs to be processed, such systems provide lower speedups than their offline counterparts. Furthermore, they do not exploit workload patterns that may speed up future queries through sample reuse. In contrast, LAQy's aggressive sample caching methodology reduces the overhead of online sampling to only online sampling for the newly created delta sample.

Hybrid AQP systems combine online and offline AQP techniques. Taster [24] proposes adapting to the workload by materializing offline samples relevant to the recent window of queries and otherwise performing online approximations. Taster makes a coarse-grained decision about full sample matches and does not explore the opportunity of partial sample reuse. LAQy exploits the overlap between required (unpredictable) and materialized (predictable) sample sets in exploratory workloads to reduce the sample creation time.

Model updates and concept drifts. The ubiquitous nature of volatile data has motivated research in machine learning and theoretical approaches to detect and update out-of-date models [9, 32, 1, 33, 4] where new data are monitored to detect when the input data distribution has significantly shifted from the maintained model. When this happens, the online model is retrained or incrementally updated.

8 Conclusion

The reuse dichotomy between the predictable predicates used by offline sampling to speculate on the useful samples and the online sampling, which pays the performance cost of unpredictability in ad-hoc queries, imposes a steep performance penalty, even if an overlapping sample exists. We propose bridging this gap by relaxing the sample reuse requirements and allowing partial sample reuse. We increase the utility of the samples created using the assumption of predictability and pay only the necessary cost to perform online sampling due to the predicate unpredictability. Our lazy sampling approach adapts to the changes in query predicates and improves performance proportionally to the reuse savings. In effect, lazy sampling allows flexible data access and computation reuse without reprocessing the full, original data input. As the data volume continues to grow, and in conjunction with modern, efficient, and hardware-conscious data management systems, to enable faster and more interactive analytical queries or speculative execution opportunities, judicious and reuse-aware sampling becomes an ever-important part of efficient approximations in modern analytical engines.

9 Acknowledgments

We thank the reviewers and the shepherd of the original SIGMOD'23 paper, and our colleagues and friends at the DIAS Lab at EPFL for their insightful comments, suggestions, and valuable feedback. This work was partially funded by the Swiss National Science Foundation (SNSF) project "Efficient Real-time Analytics on General-Purpose GPUs" (178894).

10 References

[1] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi. Mergeable summaries. In M. Benedikt, M. Krötzsch, and M. Lenzerini, editors, *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, pages 23–34. ACM, 2012.

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

[3] A. Birler, B. Radke, and T. Neumann. Concurrent online sampling for all, for free. In *Proceedings of the 16th International Workshop on Data Management on New Hardware*, DaMoN '20, New York, NY, USA, 2020. Association for Computing Machinery.

[4] M. T. Chao. A general purpose unequal probability sampling plan. *Biometrika*, 69(3):653–656, 12 1982.

[5] S. Chaudhuri, G. Das, and U. Srivastava. Effective use of block-level sampling in statistics estimation. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, page 287–298, New York, NY, USA, 2004. Association for Computing Machinery.

[6] S. Chaudhuri, B. Ding, and S. Kandula. Approximate query processing: No silver bullet. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 511–519, New York, NY, USA, 2017. Association for Computing Machinery.

[7] P. Chrysogelos, M. Karpathiotakis, R. Appuswamy, and A. Ailamaki. Hetexchange: Encapsulating heterogeneous CPU-GPU parallelism in JIT compiled engines. *Proc. VLDB Endow.*, 12(5):544–556, 2019.

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

[9] J. a. Gama, I. Žliobaitundefined, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4), mar 2014.

[10] P. B. Gibbons, V. Poosala, S. Acharya, Y. Bartal, Y. Matias, S. Muthukrishnan, S. Ramaswamy, and T. Suel. Aqua: System and techniques for approximate query answering. Technical report, Technical report, Bell Labs, 1998.

[11] H. Harmouch and F. Naumann. Cardinality estimation: An experimental survey. *Proc. VLDB Endow.*, 11(4):499–512, Dec. 2017.

[12] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig. Deepdb: Learn from data, not from queries! *Proc. VLDB Endow.*, 13(7):992–1005, Mar. 2020.

[13] S. Igescu, V. Sanca, E. Zapridou, and A. Ailamaki. Improving k-means clustering using speculation. In R. Bordawekar, C. Cappiello, V. Efthymiou, L. Ehrlinger, V. Gadepally, S. Galhotra, S. Geisler, S. Groppe, L. Gruenwald, A. Y. Halevy, H. Harmouch, O. Hassanzadeh, I. F. Ilyas, E. Jiménez-Ruiz, S. Krishnan, T. Lahiri, G. Li, J. Lu, W. Mauerer, U. F. Minhas, F. Naumann, M. T. Özsu, E. K. Rezig, K. Srinivas, M. Stonebraker, S. R. Valluri, M. Vidal, H. Wang, J. Wang, Y. Wu, X. Xue, M. Zaït, and K. Zeng, editors, *Joint Proceedings of Workshops at the 49th International Conference on Very Large Data Bases (VLDB 2023), Vancouver, Canada, August 28 - September 1, 2023*, volume 3462 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[14] S. Kandula, K. Lee, S. Chaudhuri, and M. Friedman. Experiences with approximating queries in microsoft's production big-data clusters. *Proc. VLDB Endow.*, 12(12):2131–2142, Aug. 2019.

[15] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 631–646, New York, NY, USA, 2016. Association for Computing Machinery.

[16] M. Karpathiotakis, I. Alagiannis, and A. Ailamaki. Fast queries over heterogeneous data through engine customization. *Proc. VLDB Endow.*, 9(12):972–983, 2016.

[17] A. Kim, E. Blais, A. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *Proc. VLDB Endow.*, 8(5):521–532, Jan. 2015.

[18] T. Kraska. Approximate query processing for interactive data science. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 525, New York, NY, USA, 2017. Association for Computing Machinery.

[19] K. Li and G. Li. Approximate query processing: What is new and where to go? - A survey on approximate query processing. *Data Sci. Eng.*, 3(4):379–397, 2018.

[20] Q. Ma and P. Triantafillou. Dbest: Revisiting approximate query processing engines with machine learning models. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1553–1570, New York, NY, USA, 2019. Association for Computing Machinery.

[21] P. Menon, A. Ngom, T. C. Mowry, A. Pavlo, and L. Ma. Permutable compiled queries: Dynamically adapting compiled queries without recompiling. *Proc. VLDB Endow.*, 14(2):101–113, 2020.

[22] R. B. Miller. Response time in man-computer conversational transactions. In *American Federation of Information Processing Societies: Proceedings of the AFIPS '68 Fall Joint Computer Conference, December 9-11, 1968, San Francisco, California, USA - Part I*, volume 33 of *AFIPS Conference Proceedings*, pages 267–277. AFIPS / ACM / Thomson Book Company, Washington D.C., 1968.

[23] T. Neumann. Efficiently compiling efficient query plans for modern hardware. *Proc. VLDB Endow.*, 4(9):539–550, June 2011.

[24] M. Olma, O. Papapetrou, R. Appuswamy, and A. Ailamaki. Taster: Self-tuning, elastic and online approximate query processing. In *35th IEEE*

International Conference on Data Engineering, ICDE 2019, Macao, China, April 8-11, 2019, pages 482–493. IEEE, 2019.

[25] P. O'Neil, E. O'Neil, X. Chen, and S. Revilak. *The Star Schema Benchmark and Augmented Fact Table Indexing*, page 237–252. Springer-Verlag, Berlin, Heidelberg, 2009.

[26] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. Verdictdb: Universalizing approximate query processing. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1461–1476, New York, NY, USA, 2018. Association for Computing Machinery.

[27] J. Peng, D. Zhang, J. Wang, and J. Pei. Aqp++: Connecting approximate query processing with aggregate precomputation for interactive analytics. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1477–1492, New York, NY, USA, 2018. Association for Computing Machinery.

[28] A. Raza, P. Chrysogelos, A. G. Anadiotis, and A. Ailamaki. Adaptive HTAP through elastic resource scheduling. In D. Maier, R. Pottinger, A. Doan, W. Tan, A. Alawini, and H. Q. Ngo, editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 2043–2054. ACM, 2020.

[29] V. Sanca and A. Ailamaki. Sampling-based AQP in modern analytical engines. In *DaMoN*, pages 4:1–4:8. ACM, 2022.

[30] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, VL '96, page 336, USA, 1996. IEEE Computer Society.

[31] P. Sioulas, V. Sanca, I. Mytilinis, and A. Ailamaki. Accelerating complex analytics using speculation. In *CIDR*, 2021.

[32] A. Tahmasbi, E. Jothimurugesan, S. Tirthapura, and P. B. Gibbons. Driftsurf: Stable-state / reactive-state learning under concept drift. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 10054–10064. PMLR, 2021.

[33] C. Wyman. *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX*, chapter 22, Weighted Reservoir Sampling: Randomly Sampling Streams, pages 345–349. Apress, Berkeley, CA, 2021.