Scalable Exploration of Physical Database Design

Arnd Christian König
Microsoft Research
chrisko@microsoft.com

Shubha U. Nabar
Stanford University
sunabar@stanford.edu

Abstract

Physical database design is critical to the performance of a large-scale DBMS. The corresponding automated design tuning tools need to select the best physical design from a large set of candidate designs quickly. However, for large workloads, evaluating the cost of each query in the workload for every candidate does not scale. To overcome this, we present a novel comparison primitive that only evaluates a fraction of the workload and provides an accurate estimate of the likelihood of selecting correctly. We show how to use this primitive to construct accurate and scalable selection procedures. Furthermore, we address the issue of ensuring that the estimates are conservative, even for highly skewed cost distributions. The proposed techniques are evaluated through a prototype implementation inside a commercial physical design tool.

1 Introduction

The performance of applications running against enterprise database systems depends crucially on the physical database design chosen. To enable the exploration of potential designs, today's commercial database systems have incorporated APIs that allow "What-if" analysis [8]. These take as input a query Q and a database configuration C, and return the optimizer-estimated cost of executing Q if configuration C were present. This interface is the key to building tools for exploratory analysis as well as automated recommendation of physical database designs [13, 20, 1, 10].

The problem of *physical design tuning* is then defined as follows: a physical design tool receives a *representative query workload* $\mathcal{WL}$ and constraints on the "configuration space" to explore as input, and outputs a configuration in which executing $\mathcal{WL}$ has the least possible cost (as measured by the optimizer cost model)[1]. To determine the best configuration, a number of candidate configurations are enumerated and then evaluated using "What-if" analysis.

The representative workload is typically obtained by tracing the queries that execute against a production system (using tools such as IBM Query Patroler, SQL Server Profiler, or ADDM [10]) over a representative period of time. Since a large number of SQL statements may execute in this time, the straightforward approach of comparing configurations by repeatedly invoking the query optimizer for each query in $\mathcal{WL}$ with every configuration is often not tractable [5, 20]. Our experience with a commercial physical design tool shows that a large part of the tool's overhead arises from repeated optimizer calls to evaluate large numbers of configuration/query combinations. Most of the research on such tools [13, 20, 1, 10, 7, 8] has focused on minimizing this overhead by evaluating only a few carefully chosen configurations. Our approach is orthogonal to this work and focuses on reducing the number of queries for which the optimizer calls are issued. Because the topic of which configurations to enumerate has been studied extensively, we will not comment further on the configuration-space.

Current commercial tools address the issue of large workloads by *compressing* them up-front, i.e. initially selecting a subset of queries and then tuning only this smaller set [5, 20]. These approaches do not offer any guarantees on how the compression affects the likelihood of choosing the right configuration. However, such guarantees are important due to the significant overhead of changing the physical design and the performance impact of bad database design.

The problem we study in this paper is that of *efficiently* comparing two (or more) configurations for *large* workloads. Solving this problem is crucial for scalability of automated physical design tuning tools [20, 1, 10]. We provide probabilistic guarantees on the likelihood of correctly choosing the best configuration for a large workload from a given set of candidate configurations. Our approach is to sample from the workload and use statistical inference techniques to compute the probability of selecting correctly.

The resulting probabilistic comparison primitive can be used for (a) fast interactive exploratory analysis of the configuration space, allowing the DB administrator to quickly find promising candidates for full evaluation, or (b) as the core comparison primitive inside an automated physical design tool, providing both scalability and locally good decisions with probabilistic guarantees on the accuracy of each comparison. Depending on the search strategy used, the latter can be extended to guarantees on the quality of the final result.

The key challenges to this probabilistic approach are

[1] Note that queries include both update and select statements. Thus this formulation models the trade-off between the improved performance of select-queries and the maintenance costs of additional indexes and views.

twofold. First, the accuracy of the estimation depends critically on the *variance* of the estimator we use. The challenge is thus to pick an estimator with as little variance as possible. Second, sampling techniques rely on (a) the applicability of the *Central Limit Theorem* (CLT) [9] to derive confidence statements for the estimates and (b) the sample variance being a good estimate of the true variance of the underlying distribution. Unfortunately, both assumptions may not be valid in this scenario. Thus, there is a need for techniques to determine the applicability of the CLT for the given workload and set of configurations.

1.1 Our Contributions

We propose a new probabilistic comparison primitive that – given as input a workload $\mathcal{WL}$, a set of configurations $\mathcal{C}$ and a *target probability* α – outputs the configuration with the lowest optimizer-estimated cost of executing $\mathcal{WL}$ with probability at or above the target probability α. It works by incrementally sampling queries from the original workload and computing the probability of selecting the best configuration with each new sample, stopping once the target probability is reached. Our work makes the following salient contributions:

- We derive probabilistic guarantees on the likelihood of selecting the best configuration (Section 4).
- We propose a modified sampling scheme that significantly reduces estimator variance by leveraging the fact that query costs exhibit some stability across configurations (Section 4.2).
- We show how to reduce the estimator variance further through a stratified sampling scheme that leverages commonality between queries (Section 5).
- Finally, we describe a novel technique to address the problem of highly skewed distributions in which the sample may not be representative of the overall distribution and/or the CLT may not apply for a given sample size (Section 6).

The remainder of the paper is organized as follows: In Section 2 we review related work. In Section 3 we give a formal description of the problem and introduce the necessary notation. In Section 4 we describe two sampling schemes that can be used to estimate the probability of selecting the correct configuration. In Section 5 we then show how to reduce variances through the use of stratification and combine all parts into an efficient algorithm. In Section 6 we describe how to validate the assumptions on which the probabilistic guarantees described earlier are based. We evaluate our techniques experimentally in Section 7.

2 Related Work

The techniques developed in this paper are related to the field of statistical selection and ranking [15] which is concerned with the probabilistic ranking of systems in experimental setups based on a series of measurements from each system. However, statistical ranking techniques are typically aimed at comparing systems for which the individual measurements are distributed according to a normal distribution. This is clearly not the case in our scenario. To incorporate non-normal distributions into statistical selection, techniques like *batching* (e.g. [17]) have been suggested, that initially generate a large number of measurements, and then transform this raw data into *batch means*. The batch sizes are chosen to be large enough so that the individual batch means are approximately independent and normally distributed. However, because procedures of this type need to produce a number of normally distributed estimates per configuration, they require a large number of initial measurements (according to [15], batch sizes of over 1000 measurements are common), thereby nullifying the efficiency gain due to sampling for our scenario.

Workload compression techniques such as [5, 20] compute a compact representation of a large workload before submitting it to a physical design tool. Both of these approaches are heuristics in the sense that they have no means of assessing the impact of the compression on configuration selection or consequently on physical design tuning itself. [5] poses workload compression as a clustering problem, using a distance function that models the maximum difference in cost between two queries for arbitrary configurations. This distance function does not use the optimizer's cost estimate, so it is not clear how well this approximation holds for complex queries. [20] compresses a workload by selecting queries in order of their current costs, until a user-specifiable percentage X of the total workload cost has been reached. While computationally simple, this approach may lead to a significant reduction in tuning quality for workloads in which queries of some templates are generally more expensive than the remaining queries, as then only some templates are considered for tuning. Making X user-specifiable does not alleviate this problem, for the user has no way of assessing the impact of X on tuning quality.

The most serious drawback of both approaches is that they do not adapt the number of queries retained to the space of configurations considered. Consequently, they may result in compression that is either too conservative, resulting in excessive numbers of optimizer calls or too coarse, resulting in inferior physical designs.

3 Problem Statement

In this section we give a formal definition of the problem addressed in this paper. This definition is identical to the informal one given in Section 1.1, but in addition to the parameters $\mathcal{C}, \mathcal{WL}$ and the target probability α, we introduce an additional parameter δ, which describes the minimum difference in cost between configurations that we care to detect. Specifying a value of $\delta > 0$ helps to avoid scenarios in which the algorithm samples a large fraction of the workload when comparing configurations of (nearly) identical costs. While accuracy for such configurations is necessary in some cases, detecting large differences is all that matters in others. For instance, when comparing a new candidate configuration to the current one, the overhead of changing the physical database design is justified only when the new configuration is significantly better.

3.1 Notation

In this paper, we use $Cost(q, C)$ to denote the optimizer-estimated cost of executing a query q in a configuration C. Similarly, the total estimated cost of executing a set of queries $\{q_1, \ldots, q_N\}$ in configuration C is $Cost(\{q_1, \ldots, q_N\}, C) := \sum_{i=1}^{N} Cost(q_i, C)$; when the set of queries includes the entire workload, this will be referred to as 'the cost of configuration C'. In the interest of simple notation, we will use the simplifying assumption that the overhead of making a single optimizer call is constant across queries[2]. In the remainder of the paper we use the term 'cost' to describe optimizer-estimated query costs.

We use the phrase 'to sample a query' to denote the process of obtaining the query's text from a workload table or file, and evaluating its cost using the query optimizer. The configuration being used to evaluate the query will be clear from context.

The probability of an event A will be denoted by $Pr(A)$, and the conditional probability of event A given event B by $Pr(A|B)$.

3.2 The Configuration Selection Problem

Problem Formulation: *Given a set of physical database configurations $\mathcal{C} = \{C_1, \ldots, C_k\}$ and a workload $\mathcal{WL} = \{q_1, \ldots, q_N\}$, a target probability α and a sensitivity parameter δ, select a configuration C_i such that the probability of a correct selection, $Pr(CS)$, is larger than α, where* $Pr(CS) :=$

$$Pr(Cost(\mathcal{WL}, C_i) < Cost(\mathcal{WL}, C_j) + \delta, \forall j \neq i), \quad (1)$$

while making a minimal number of optimizer-calls.

Our approach to solving the configuration selection problem is to repeatedly sample queries from $\mathcal{WL}$, evaluate $Pr(CS)$, and terminate once the target probability α has been reached. Next, we describe how to estimate $Pr(CS)$ (Section 4) and how to use these estimates and stratification of the workload to construct an algorithm for the configuration selection problem (Section 5).

4 Sampling Techniques

In this section we present two sampling schemes to derive $Pr(CS)$ estimates – first we describe a straightforward approach called *Independent Sampling* (Section 4.1); then we describe *Delta Sampling*, which exploits properties of query costs to come up with a more accurate estimator (Section 4.2).

4.1 Independent Sampling

Independent Sampling is the base sampling scheme that we use to estimate the differences in costs of pairs of configurations. First, we define an unbiased estimator X_i of $Cost(\mathcal{WL}, C_i)$. The estimator is obtained by sampling a set of queries $\mathcal{SL}_i$ from $\mathcal{WL}$ and is calculated as

$$X_i = \frac{N}{|\mathcal{SL}_i|} \sum_{q \in \mathcal{SL}_i} Cost(q, C_i),$$

i.e., X_i is the mean of $|\mathcal{SL}_i|$ random variables, scaled up by the total number of queries in the workload. The variance of the underlying cost-distribution is

$${\sigma_i}^2 = \frac{\sum_{l=1}^{N} (Cost(q_l, C_i) - \frac{Cost(\mathcal{WL}, Ci)}{N})^2}{N}.$$

Now, when using a simple random sample of size n to estimate X_i, the variance of this estimate is $Var(X_i) := \frac{N^2}{n} \frac{{\sigma_i}^2 \cdot N}{N-1} \left(1 - \frac{n}{N}\right)$ [9]. In the following, we will use the shorthand ${S_i}^2 := {\sigma_i}^2 \cdot N/(N-1)$.

To choose the better of two configurations C_l and C_j we now use the random variable $X_l - X_j$, which is an unbiased estimator of the true difference in costs $\mu_{l,j} := Cost(\mathcal{WL}, C_l) - Cost(\mathcal{WL}, C_j)$. Under large-sample assumptions, the standardized random variable

$$\Delta_{l,j} := \frac{(X_l - X_j) - \mu_{l,j}}{N \sqrt{\left(\frac{{S_l}^2}{|\mathcal{SL}_l|} \left(1 - \frac{|\mathcal{SL}_l|}{N}\right) + \frac{{S_j}^2}{|\mathcal{SL}_j|} \left(1 - \frac{|\mathcal{SL}_j|}{N}\right) \right)}} \quad (2)$$

is normally distributed with mean 0 and variance 1 (according to the CLT). Based on this we can assess the probability of making a correct selection when choosing between C_l and C_j, which we write as $Pr(CS_{l,j})$.

The decision procedure to choose between the two configurations is to pick the configuration C_l, for which the estimated cost X_l is the smallest. In this case the probability of making an *incorrect* selection corresponds to the event $X_l - X_j < 0$ and $\mu_{l,j} > \delta$. Thus,

$$\begin{aligned} Pr(CS_{l,j}) &= 1 - Pr(X_l - X_j < 0 | \mu_{l,j} > \delta) \\ &= Pr(X_l - X_j \geq 0 | \mu_{l,j} > \delta) \\ &\geq Pr(X_l - X_j \geq 0 | \mu_{l,j} = \delta) = \end{aligned}$$

$$Pr\left(\Delta_{l,j} > \frac{-\delta}{N \sqrt{\frac{{S_l}^2}{|\mathcal{SL}_l|} \left(1 - \frac{|\mathcal{SL}_l|}{N}\right) + \frac{{S_j}^2}{|\mathcal{SL}_j|} \left(1 - \frac{|\mathcal{SL}_j|}{N}\right)}} \right).$$

[2] We discuss how to incorporate differences in optimization costs between queries in Section 5.2.

Given that $\Delta_{l,j} \sim N(0,1)$, we can compute this probability using the appropriate lookup-tables. Because the true variances of the cost distributions are not known, we use the sample variances

$$s_i^2 = \frac{\sum_{q \in \mathcal{SL}_i} (Cost(q,C_i) - \frac{\sum_{q \in \mathcal{SL}_i} (Cost(q,C_i))}{|\mathcal{SL}_i|})^2}{|\mathcal{SL}_i| - 1},$$

which are unbiased estimators of the true variances, in their place in $Pr(CS_{l,j})$. Consequently, accurate estimation of $Pr(CS_{l,j})$ depends on (a) having sampled sufficiently many values for the *Central Limit Theorem* (CLT) [9] to be applicable and (b) being able to estimate σ_i^2 accurately by s_i^2. It is not possible to verify these conditions based on a sample alone, as a single very large outlier value may dominate both the variance and the skew of the cost distribution. We will describe how to use additional domain knowledge on the cost distributions to verify (a) and (b) in Section 6. In practice, a standard rule-of-thumb is to sample $n_{min} := 30$ queries before $Pr(CS)$ is computed based on the normality of $\Delta_{l,j}$. While this in many cases results in the CLT being applicable, the dependence on the sample-variances s_i^2 still means that $Pr(CS)$ may be either over- or under-estimated.

When the number of configurations k is greater than two, we derive the probability of correct selection $Pr(CS)$ based on the pairwise selection probabilities $Pr(CS_{l,j})$. Once again, the selection procedure here is to choose the configuration, C_i, with the smallest estimate, X_i. Consider the case where C_i is chosen as the best configuration from $C_1, \ldots, C_k$. C_i would be an incorrect choice, if for any $j \neq i$, $X_i - X_j < 0$ but $\mu_{i,j} > \delta$. Thus we can derive a trivial bound for $Pr(CS)$ using the *Bonferroni inequality*:

$$Pr(CS) \geq \quad 1 - \sum_{j \in \{1,\ldots,k\}, j \neq i} \left(1 - Pr(CS_{i,j})\right). \quad (3)$$

4.2 Delta Sampling

Delta-Sampling is a variation of Independent Sampling, that leverages the stability of the rankings of query costs across configurations to obtain significant reduction in the variance of the resulting estimator. In Delta Sampling, we pick only a single set of queries $\mathcal{SL} \subseteq \mathcal{WL}$ from the workload and then estimate the difference in total cost between two configurations C_l, C_j as

$$X_{l,j} = \frac{N}{|\mathcal{SL}|} \sum_{q \in \mathcal{SL}} (Cost(q,C_l) - Cost(q,C_j)).$$

The selection procedure here is to pick the configuration C_l for which $X_{l,j} < 0$. To give bounds for the probability of correct selection, we define

$$\Delta_{l,j} := \frac{X_{l,j} - \mu_{l,j}}{N \cdot \sqrt{\frac{S_{l,j}^2}{|\mathcal{SL}|} \left(1 - \frac{|\mathcal{SL}|}{N}\right)}} \quad (4)$$

where $S_{l,j}^2 := \frac{N}{N-1} \cdot \sigma_{l,j}^2$ and $\sigma_{l,j}^2$ is the variance of the distribution $\{Cost(q_i, C_l) - Cost(q_i, C_j) | i = 1, \ldots, N\}$ of the cost-differences between C_l and C_j. Again $\Delta_{l,j} \sim N(0,1)$ and we can derive $PR(CS_{l,j})$ as in the case of Independent Sampling. The lower bound on $PR(CS)$ from equation 3 in the case of multiple configurations applies here as well.

To explain why Delta Sampling typically outperforms Independent sampling, compare equation 4 to equation 2. The difference in estimator-variance between Delta Sampling and Independent Sampling depends on the difference between the terms $S_{l,j}^2/|\mathcal{SL}|$ (in Equation 4) and $S_l^2/|\mathcal{SL}_l| + S_j^2/|\mathcal{SL}_j|$ (in Equation 2), i.e. on the weighted difference between the respective variances. It is known that $\sigma_{l,j}^2 = \sigma_l^2 + \sigma_j^2 - 2 \cdot Cov_{l,j}$ [3], where $Cov_{l,j}$ is the covariance of the distributions of the query costs of the workload when evaluated in configurations C_l and C_j.

Delta Sampling exploits this as follows. For large workloads it typically holds that – when ranking the queries by their costs in C_l and C_j – the ranks for each individual query do not vary too much between configurations (especially configurations of similar costs which correspond to difficult selection problems). This means that in most cases if the cost of a query q_i, when evaluated in C_l is higher than the average cost of all queries in C_l, its cost when evaluated in C_j is likely to be higher than that of the average query in C_j (and vice versa for below-average costs). For example, multi-join queries will be typically more expensive than single-value lookups, no matter what the physical design. Now, the covariance $Cov_{l,j}$ is defined as

$$\sum_{i=1}^{N} \left(\left(Cost(q_i, C_l) - \frac{Cost(\mathcal{WL}, C_l)}{N}\right) \left(Cost(q_i, C_j) - \frac{Cost(\mathcal{WL}, C_j)}{N}\right) \right).$$

All queries q_i for which the above property holds, will add a positive term to the summation in the above formula (as the signs of both factors will be equal). Consequently, if this property holds for sufficiently many queries, the covariance $Cov_{l,j}$ itself will be positive, thereby making $\sigma_{l,j}^2 < \sigma_l^2 + \sigma_j^2$. So, if the number of optimizer calls are roughly equally distributed between configurations in Independent Sampling, Delta-Sampling should result in tighter estimates of $Pr(CS)$. We verify this experimentally in Section 7.

5 Workload Stratification

It is possible to further reduce the estimator variances using *stratified sampling* [9]. Stratified sampling is a generalization of uniform sampling, in which the workload is partitioned into L disjunct 'strata' $\mathcal{WL}_1 \cup \ldots \cup \mathcal{WL}_L = \mathcal{WL}$. Strata that exhibit high variances contribute more samples.

We introduce stratification by partitioning the $\mathcal{WL}$ into increasingly fine strata as the sampling progresses. The resulting algorithm for the configuration selection problem is shown in Algorithm 1. The algorithm samples from $\mathcal{WL}$ until $Pr(CS)$ is greater than α. After each sample, we check if further stratification is expected to lower estimator variance; if so, we implement it. We describe this in detail in Section 5.1. Once the workload is stratified, we also have to choose which stratum to pick the next sample from. We describe this in detail in Section 5.2.

A further heuristic (for large k) is to stop sampling for configurations that are clearly not optimal. If C_l is the chosen configuration in an iteration and $1 - Pr(CS_{l,j})$ is negligible for some C_j, then C_j is dropped from future iterations. This way, many configurations are eliminated quickly, and only the ones contributing the most uncertainty remain in the pool. We study the effects of this optimization in Section 7.

```
// Input: Workload $\mathcal{WL}$, Configurations $C_1, \ldots, C_k$,
//           Probability $\alpha$, Sensitivity $\delta$
// Output: Configuration $C_{best}$, Probability $Pr(CS)$
For each $C_l$, pick pilot-sample $\mathcal{SL}_l, |\mathcal{SL}_l| = n_{min}$.
// Note: for Delta-Sampling $\mathcal{SL} = \ldots = \mathcal{SL}_k$
repeat
    if further Stratification beneficial then
        Create new strata; sample additional queries so that
        every stratum contains $\geq n_{min}$ queries (Section 5.1)
    end if
    Select next query $q$ and – in case of Independent Sampling
    – configuration $C_i$ to evaluate(Section 5.2)
    Evaluate $\Delta_{i,j}$ for all $1 \leq i < j \leq k$ (Sections 4.1, 4.2)
    Select $C_{best}$ from $\mathcal{C}$ s.t. $X_{best} - X_j < 0 \; \forall j \neq best$
    for Independent-sampling or $X_{best,j} < 0 \; \forall j \neq best$ for
    Delta-sampling (Sections 4.1, 4.2)
until $Pr(CS) > \alpha$
return $C_{best}$,$Pr(CS)$
```

Algorithm 1: Computing Pr(CS) through Sampling

Preprocessing: For workloads large enough that the query strings do not fit into memory, we write all query strings to a database table, which also contains the query's ID and *template* (also known as a query's *signature* [6, 5] or *skeleton* [18]). Two queries have the same template if they are identical in everything but the constant bindings of their parameters. Template-information can be acquired either by using a workload-collection tool capable of recording template information (e.g. [6]) or by parsing the queries, which generally requires a small fraction of the overhead necessary to optimize them. Now we can obtain a random sample of size n from this table by computing a random permutation of the query IDs and then (using a single scan) reading the queries corresponding to the first n IDs into memory. This approach trivially extends to stratified sampling.

5.1 Progressive Stratification

Given the strata $\mathcal{WL}_h, h = 1, \ldots, L$ and a configuration C_i, the stratified estimates X_i, in the case of independent sampling, are calculated as the sum of the weighted estimates for each stratum $\mathcal{WL}_h$. Let the variance of the cost distribution of $\mathcal{WL}_h$, when evaluated in configuration C_i be denoted by $\sigma^2_{i(h)}$ (and $S^2_{i(h)} := \sigma^2_{i(h)} \cdot \frac{|\mathcal{WL}_h|}{(|\mathcal{WL}_h|-1)}$). Let the number of queries sampled from $\mathcal{WL}_h$ be n_h. Then the variance of the estimator X_i of the cost of C_i for Independent Sampling is [9]

$$Var(X_i) = \sum_{h=1}^{L} |\mathcal{WL}_h|^2 \frac{S^2_{i(h)}}{n_h} \cdot \left(1 - \frac{n_h}{|\mathcal{WL}_h|}\right). \qquad (5)$$

We would thus like to pick a stratification and an allocation of sample sizes $n_1 + \ldots + n_L = n$ that minimizes equation 5. For Delta-sampling, the variances of the estimators $X_{i,j}$ can be obtained analogously.

Picking a stratification: Initially, consider the case of Independent Sampling: here, we can use a different stratification of $\mathcal{WL}$ for every X_i. In the following we discuss how to stratify $\mathcal{WL}$ for one configuration C_i.

Because the costs of queries that share a template typically exhibit much smaller variance than the costs of the entire workload, it is often possible to get a good estimate of the average cost of queries sharing a template using only few sample queries. Consequently, we only consider stratifications in which all queries of one template are grouped into the same stratum and use the average costs per template to estimate the strata variances $S^2_{i(h)}$. This does not mean that using a very fine-grained stratification with a single stratum for every template must necessarily be the best way to stratify. While a fine stratification may result in a small variance for large sample-sizes, it comes at a price: in order for the estimate based on stratified sampling to be valid, the number of samples n_h taken from each stratum $\mathcal{WL}_h$ has to be sufficiently large so that the estimator of the cost of a configuration in *each* stratum is normal.

Choosing between stratifications: Based on the above, we can compute the stratum variances resulting from a stratification $\mathcal{ST} = \{\mathcal{WL}_1, \ldots, \mathcal{WL}_L\}$, given a sample allocation $\mathcal{NT} = (n_1, \ldots, n_L)$ using equation 5. Using these variances, we can then estimate the number of samples, $\#Total\_Samples$, necessary to achieve $Pr(CS) > \alpha$ for a set of configurations $\mathcal{C}$. We omit the details of this algorithm due to space constraints, but it essentially involves computing target variances $TargetVar(X_i)$ for each estimator X_i such that these variances, when substituted in to the formulas in Section 4, give us the target probability. For a given a stratification $\mathcal{ST}$, initial sample allocation $\mathcal{NT}$ and configuration C_i we denote the minimum number of samples required to achieve the target variance (under the assumption that the underlying $\sigma^2_{i(h)}$ remain constant) by $\#Samples(C_i, \mathcal{ST}, \mathcal{NT})$.[3] We use this estimate to compare different stratification schemes.

We now choose the stratification of $\mathcal{WL}$ for a configuration C_i as follows: when sampling, we maintain the average cost of the queries for C_i in each template and use it to estimate $S^2_{i(h)}$, once we have seen a small number of queries for each template. The algorithm starts with a single stratum ($\mathcal{ST} = \mathcal{WL}$). After every sample from C_i it evaluates

[3] If we ignore the finite population correction, then this number can be computed using $O(L \cdot \log_2(N))$ operations by combining a binary search and *Neyman Allocation* [9].

if it should change the stratification of the queries for this configuration. This is done by iterating over a set of possible new stratifications $\mathcal{ST}_p, p = 1\ldots,\#templates - 1$ – resulting from splitting one of the existing strata into two at the pth-most expensive template – and computing $\#Samples(C_i, \mathcal{ST}_p,\ \mathcal{NT}_p)$. Here, we chose the $\mathcal{NT}_p$ such that initially each stratum contains the maximum of the number of queries already sampled from it and n_{min}. For scalability, we only add a single stratum per step. Note that unlike stratification in the context of survey sampling or approximate query processing (e.g. [4] where the chosen stratification is the result of a complex optimization problem), we incur negligible computational overhead for changing the stratification. The details are shown in Algorithm 2.

```
// Input: Strata $\mathcal{WL}_1, \ldots, \mathcal{WL}_L$, Configuration $C_i$
// Output: Stratum to split $\mathcal{WL}_s = \mathcal{WL}_s^1 \cup \mathcal{WL}_s^2$
//          Templates to store in $\mathcal{WL}_s^1, \mathcal{WL}_s^2$
$s := L + 1$. // Initialization
$min\_sam :=$
    $\#Samples(C_i, \{\mathcal{WL}_1, .., \mathcal{WL}_L\}, (n_1, .., n_L))$.
for all $\mathcal{WL}_j \in \{\mathcal{WL}_1, \ldots, \mathcal{WL}_L\}$ do
    Set $n'_j :=$ the expected number of samples from $\mathcal{WL}_j$ in
    $\#Samples(C_i, \{\mathcal{WL}_1, \ldots, \mathcal{WL}_L\}, (n_1, \ldots, n_L))$.
    if $n'_j \geq 2 \cdot n_{min}$ then
        Order the query templates in $\mathcal{WL}_j$ by their average cost
        as $tmp_{p_1}, \ldots, tmp_{p_m}$.
        for Every split point $t \in \{p_1, \ldots, p_{m-1}\}$ do
            Consider the split $\mathcal{WL}_j = \mathcal{WL}_j^1 \cup \mathcal{WL}_j^2$ into tem-
            plates $\{tmp_{p_1}, \ldots, tmp_t\}$, $\{tmp_{t+1}, \ldots, tmp_{p_m}\}$.
            Compute $sam[t] :=$
            $\#Samples(C_i, \{\mathcal{WL}_1 \ldots \mathcal{WL}_j^1, \mathcal{WL}_j^2 \ldots \mathcal{WL}_L\},$
            $(n_1, \ldots, \max\{n_{min}, |\mathcal{WL}_j^1|\}, \max\{n_{min}, |\mathcal{WL}_j^2|\}$
            $, \ldots, n_L))$.
            if $sam[t] < min\_sam$ then
                $min\_sam := sam[t]; s = t$.
            end if
        end for
    end if
end for
if $s \neq L + 1$ then
    Update $\#Total\_Samples$ and target variances.
    return $\mathcal{WL}_s$ and template information.
end if
```

Algorithm 2: Computing the optimum stratification

Overhead: Given T templates in $\mathcal{WL}$, there can only be $T - 1$ possible split points over all strata. For each split point, we have to evaluate the corresponding value of $\#Samples(\ldots)$, which requires $O(L \cdot \log_2(N))$ operations. Consequently, the entire algorithm runs in $O(L \cdot \log_2(N) \cdot T)$ time. Because the number of different templates is typically orders of magnitude smaller than the size of the entire workload, the resulting overhead is negligible when compared to the overhead of optimizing even a single query.

Note that we execute this algorithm only for the configuration C_i that the last sample was chosen from, as the estimates and sample variances for the remaining configurations stay unchanged.

Stratification for Delta Sampling: For Delta-Sampling, the problem of finding a good stratification is complicated by the fact that Delta-Sampling samples the same set of queries for all configurations and thus a stratification scheme needs to reconcile the variances of multiple random variables $X_{i,j}$. As a consequence, we consider the reduction in the average variance of $X_{i,j}, 1 \leq i < j \leq k$ when comparing stratification schemes. Furthermore, the ordering of the templates for each $X_{i,j}$ may vary, resulting in up to $k \cdot (k-1)/2$ orderings to consider when computing a new candidate stratification. For reasons of tractability, we only consider a single ranking over the $X_{i,j}$ instead.

5.2 Picking the next sample

We first consider Independent Sampling without stratification: ideally, we would like to pick the next combination of query and configuration to evaluate so that $Pr(CS)$ is maximized. We use a heuristic approach, attempting to minimize the sum of the estimator-variances instead. In case of Independent Sampling this means increasing one sample-size $|\mathcal{SL}_j|$ by one, so that $\sum_{i=1}^{k} Var(X_i)$ is minimized. Since we don't know the effect the next query will have on sample means and ${s_i}^2$s beforehand, we estimate the change to the sum of variances assuming that these remain unchanged. For Delta-Sampling, the sampled query is evaluated in every configuration, so sample selection is trivial.

If the workload is stratified, we use a similar approach for Independent Sampling, choosing the configuration and stratum resulting in the biggest estimated improvement in the sum of variances. For Delta-Sampling with stratification, a query from the stratum resulting in the biggest estimated improvement in the sum of the variances of all estimators is chosen and evaluated in every configuration.

While this approach assumes optimization times to be constant, different optimization times for each template can be modeled by computing the average overhead for each configuration/stratum pair and selecting the one maximizing the variance reduction relative to the expected overhead.

6 Applicability of the CLT

As discussed in Section 4.1, the estimation of $Pr(CS)$ relies on the fact that (i) we have sampled sufficiently many queries for the central limit theorem to apply and (ii) the sample variances ${s_i}^2$ are accurate estimators of the true variances ${\sigma_i}^2$. While these assumptions often do hold in practice, the potential impact of choosing an inferior database design makes it desirable to be able to validate them.

We do this as follows: in the scenario of physical database design, we do know the total number of queries in the workload and can obtain upper and lower bounds on their individual costs. Using these, we then compute upper bounds on the skew and variance of the underlying distribution and use these to verify the assumptions (i) and (ii). In

Section 6.1, we will describe how to obtain bounds on the costs of queries that have not been sampled; then, in Section 6.2 we describe how to use these bounds to compute upper bounds on skew and variance.

6.1 Deriving Cost Bounds for Queries

The problem of bounding the true cost of arbitrary database queries has been studied extensively and is known to be hard (e.g. [14]). However, in the context of physical database design, we only seek the configuration with the best *optimizer-estimated* cost, and not bounds on the true selectivity of query expressions. This simplification allows us to utilize knowledge on the space of physical database designs and the optimizer itself to make this problem tractable; moreover, it is actually essential to physical design tuning, as the tuning tool cannot influence optimizer-decisions at run-time and thus needs to keep its recommendations in sync with optimizer behavior.

First, consider the issue of obtaining bounds on SELECT-queries. In the context of automated physical design tools, it is possible to determine a so-called *base configuration*, consisting of all indexes and views that will be present in all configurations enumerated during the tuning process. If the optimizer is well-behaved, then adding an index or view to the base configuration can only improve the optimizer estimated cost of a SELECT-query. Consequently, the cost of a SELECT-query in its base configuration gives an upper bound on the cost for any configuration enumerated during the tuning. Lower bounds on the costs of a SELECT-query can be obtained using the reverse method: using the cost of a query Q in a configuration containing all indexes and views that may be useful to Q. All automated physical design tools known to us have components that suggest a set of structures the query may benefit from; however, ultimately the number of such structures may be very large. To overcome this, it is possible to use the techniques described in [2]. Here, the query-optimizer is instrumented with additional code that outputs – for any individual access path considered during the optimization of a query – the corresponding index or view that would be optimal to support this access path. This reduces the number of relevant indexes and views significantly, as (i) we do not have to 'guess' which access paths may be relevant and (ii) while a complex SQL query may be executed in a extremely large number of different ways, the optimizer itself – for reasons of scalability – only considers a subset of them.

In order to bound the costs of UPDATE statements (including INSERT and DELETE statements), we use the standard approach of splitting a complex update statement into a SELECT and an UPDATE part, e.g. the statement 'UPDATE R SET $A_1 = A_3$ WHERE $A_2 < 4$' is separated into:

(i) SELECT A_3 FROM R WHERE $A_2 < 4$, and
(ii) UPDATE TOP(k) R SET $A1 = 0$,

where k is the estimated selectivity of (i). The SELECT-part of the query is handled as outlined above. To bound the cost of the UPDATE-part, we use two observations: (1) the number of different update templates occurring tends to be orders of magnitude smaller than the workload size and (2) in the cost models currently used in query optimizers, the cost of a pure update statement grows with its selectivity. Thus, we can bound the cost of all UPDATE-statements of a specific template T on a configuration C using the optimizer cost-estimate in C for the two queries with the largest/smallest selectivity in T. This means that we have to make 2 optimizer calls per template and configuration (unlike when bounding the cost of SELECT queries); however, this approach scales well since the number of different templates is generally orders of magnitude smaller than the size of the workload. Note that automated physical design tuning tools already issue a single optimizer call for all queries in $\mathcal{WL}$ to derive initial costs (e.g. [20], figure 2) and require a second optimizer call per query for report generation.

While these techniques are very simple, even very conservative cost bounds tend to work well, as the resulting confidence bounds given by the CLT converge quickly. An exhaustive discussion of bounding techniques for SQL queries is not the focus of this paper, and these are an interesting research challenge themself.

6.2 Verifying the validity of the estimators

Now, the presence of bounds on the costs of the queries not part of the sample allows us to compute an upper bound σ^2_{max} on ${\sigma_i}^2$, which we can use in its place, thereby ensuring that the estimated value of $Pr(CS)$ will be conservative. Computing σ^2_{max} is a maximization problem with constraints defined by the cost intervals.

Problem Statement [Bounding the Variance]: *Given a set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$ representing query costs, with each v_i bounded by $low_i \leq v_i \leq high_i$, compute*

$$\sigma^2_{max} = \max_{\substack{(v_1,\ldots,v_n)\in\mathbb{R}^n \\ \forall i: low_i \leq v_i \leq high_i}} \frac{1}{n} \sum_{i=0}^{n} \left(v_i - \frac{\sum_{i=0}^{n} v_i}{n} \right)^2. \quad (6)$$

It is known that this problem is NP-hard [11] to solve exactly or to approximate within an arbitrary $\epsilon > 0$ [12]. The best known algorithm for this problem requires up to $O(2^n \cdot n^2)$ operations [12], and is thus not practical for the workload sizes we consider. Therefore, we propose a novel algorithm approximating σ^2_{max}. The basic idea is to solve the problem for values of v_i that are multiples of an appropriately chosen factor ρ and to bound the difference between the solution obtained this way (which we'll refer to as $\hat{\sigma}^2_{max}$) and the solution σ^2_{max} for unconstrained v_i. We use the notation v_i^ρ when describing the rounded values. Rewriting equation 6, we get

$$\sigma^2_{max} = \max_{\substack{(v_1,\ldots,v_n)\in\mathbb{R}^n \\ \forall i: low_i \leq v_i \leq high_i}} \frac{1}{n} \left(\sum_{i=0}^{n} (v_i)^2 - n \left(\frac{1}{n} \sum_{i=0}^{n} v_i \right)^2 \right). \quad (7)$$

Now, we define $MaxV^2[m][j]$ as the maximum value of $\sum_{i=1}^{m}(v_i^\rho)^2$ under the constraint that $\sum_{i=1}^{m}(v_i^\rho) = \sum_{i=1}^{m} low_i^\rho + j \cdot \rho$. Consequently, every solution $\hat{\sigma}^2_{max}$ to the maximization problem for multiples of ρ must be of the form

$$\frac{1}{n}\big(MaxV^2[n][j] - n \cdot (\frac{1}{n}\sum_{i=1}^{m} low_i^\rho + j \cdot \rho)^2\big), \quad (8)$$

and we can find the solution by examining all possible values of equation 8. Because we are only considering multiples of ρ, we round the boundaries of each v_i to the closest multiples of ρ: $low_i^\rho := \lfloor (low_i + \rho/2)/\rho \rfloor \cdot \rho$ and $high_i^\rho := \lfloor (high_i + \rho/2)/\rho \rfloor \cdot \rho$. Thus each v_i can only take on $range_i = (high_i^\rho - low_i^\rho)/\rho + 1$ distinct values. Consequently, the average of the first m values can take on $total_m = \big(\sum_{i=1}^{m} range_i\big) - (m-1)$ different values. This limitation is the key idea to making our approximation algorithm scalable, as the values of $total_i$ typically increase much more slowly than the number of combinations of different $high_i$ and low_i values.

We can compute all possible values of $MaxV^2[m][j]$ for $m = 1, \ldots, n, j = 0, \ldots, total_{m-1} - 1$ using the following recurrence:

$$MaxV^2[i][j+l]$$

$$= \begin{cases} \max\limits_{l=1,\ldots,total_{i-1}} MaxV^2[i-1][l] + (low_i^\rho + j \cdot \rho)^2, \text{if } i > 1 \\ (low_i^\rho + j \cdot \rho)^2, \text{otherwise.} \end{cases}$$

This computation requires $total_{i-1}$ steps for each $i = 1, \ldots, n$. Now we can compute the solution to the constrained maximization problem using equation 8 in $total_n$ steps. Two further performance-optimizations are possible: because the 2nd central moment of a distribution has no global maximum over a compact box that is not attained at a boundary point [16], each v_i^ρ must either take on its maximum or minimum value. Consequently, we only need to check the cases of $j = 0$ and $j = total_{m-1} - 1$ in the recurrence. Furthermore, we can minimize the number of steps in the algorithm by traversing the values v_i in increasing order of $range_i$ when computing $MaxV^2[m][j]$.

Accuracy of the approximation: The difference between the approximate solution $\hat{\sigma}^2_{max}$ and the true optimum σ^2_{max} can be bounded as follows: the existence of a set of variables $v_1, \ldots, v_n$ with $\forall i : low_i \leq v_i \leq high_i$ implies that there exists a set of of multiples of ρ, $v_1^\rho, \ldots, v_n^\rho$ with $\forall i : low_i^\rho \leq v_i^\rho \leq high_i^\rho$, so that $\forall i : |v_i - v_i^\rho| \leq \rho/2$. Consequently, the difference between the value of $\sum_{i=0}^{n}(v_i^\rho)^2$ and the value of $\sum_{i=0}^{n}(v_i)^2$ in equation 7 for these sets of values is at most $\sum_{i=0}^{n}\big(\rho \cdot v_i^\rho + \rho^2/4\big)$. Similarly, the difference between $n \cdot (\frac{1}{n}\sum_{i=0}^{n} v_i^\rho)^2$ and $n \cdot (\frac{1}{n}\sum_{i=0}^{n} v_i)^2$ in equation 7 is at most $\sum_{i=0}^{n}\big(\rho \cdot v_i^\rho + \rho^2/4\big)$.

Therefore, the existence of a solution σ^2_{max} to equation 6 implies the existence of a set of of multiples of ρ, $v_1^\rho, \ldots, v_n^\rho$ with $\forall i : low_i^\rho \leq v_i^\rho \leq high_i^\rho$ such that the difference between σ^2_{max} and the variance of $v_1^\rho, \ldots, v_n^\rho$ is bounded by $\theta := \frac{2}{n}\sum_{i=0}^{n}\big(\rho \cdot v_i^\rho + \rho^2/4\big)$ (*).

Similarly, the existence of a solution $\hat{\sigma}^2_{max}$ to the optimization problem for multiples of ρ implies the existence of a set of variables $v_1, \ldots, v_n$ with $\forall i : low_i \leq v_i \leq high_i$ such that the difference between $\hat{\sigma}^2_{max}$ and the variance of $v_1, \ldots, v_n$ is bounded by θ as well (**).

Therefore, if we compute a solution $\hat{\sigma}^2_{max}$ to the optimization problem for multiples of ρ, then (**) implies that the solution to the unconstrained problem is at least $\hat{\sigma}^2_{max} - \theta$ and (*) implies that no solution σ^2_{max} to equation 6 exists that is larger than $\hat{\sigma}^2_{max} + \theta$.

Overhead: To demonstrate the scalability of this approximation algorithm, we have measured the overhead of computing $\hat{\sigma}^2_{max}$ for a TPC-D workload of 100K queries and various values of ρ on a Pentium 4 (2.8GHz) PC in Table 1.

	$N{=}100K$ $\rho{=}10$	$N{=}100K$ $\rho{=}1$	$N{=}100K$ $\rho{=}1/10$
Time($\hat{\sigma}^2_{max}$)	0.4 sec.	5.2 sec.	53 sec.

Table 1: Overhead of approximating σ^2_{max}

Verifying the applicability of the CLT: The 2nd assumption made when modeling $Pr(CS)$ is that the sample size is sufficiently large for the CLT to apply. However, we have not stated thus far how a sufficiently large minimum sample size n_{min} is picked. One general result in this area is *Cochran's rule* ([9], p. 42) which states that for populations marked by positive skewness (i.e. the population contains significant outliers), the sample size n should satisfy $n > 25 \cdot (G_1)^2$, where G_1 is Fisher's measure of skew. Under the assumption that the disturbance in any moment higher than the 3rd is negligible (and additional conditions on the sampling fraction $f := 1 - n/N$), it can be shown that this condition guarantees that a 95% confidence statement will be wrong no more than 6% of the time, i.e. for the standardized statistic $Z_n := \sqrt{\frac{n}{1-f}\frac{X_i - \mu_i}{\sqrt{\sigma^2}}}$:

$$Pr(Z_n \leq 1.96) - Pr(Z_n \leq -1.96) > 0.94.$$

The derivation of additional results of this type is studied in [19]. While details of this work are beyond the scope of this paper, we use a modification of Cochran's Rule proposed in [19], which was found to be robust for the population sizes we consider:

$$n > 28 + 25 \cdot (G_1)^2. \quad (9)$$

In order to verify this condition, we need to be able to compute an upper bound on G_1:

Problem Statement [Bounding the skew]: *Given a set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$, with each v_i bounded by*

$low_i \leq v_i \leq high_i$, *compute*

$$G1_{max} = \max_{\substack{(v_1,\ldots,v_n)\in\mathbb{R}^n \\ \forall i: low_i \leq v_i \leq high_i}} \frac{\sum_{i=0}^{n}\left(v_i - \frac{\sum_{i=0}^{n} v_i}{n}\right)^3}{n \cdot \left(\sum_{i=0}^{n}\left(v_i - \frac{\sum_{i=0}^{n} v_i}{n}\right)^2\right)^{\frac{3}{2}}}.$$

Unlike the case of variance maximization, to the best of our knowledge, the complexity of maximizing G_1 is not known. We approach this problem using an approximation scheme similar to the one used for σ^2_{max}; because of size constraints, we omit the full description of this algorithm.

Because of the quick convergence of the CLT, even the rough bounds derived in the previous section allow us to formulate practical constraints on sample size. For example, for the highly skewed (query costs vary by multiple degrees of magnitude) $13K$ query TPC-D workload described in Section 7, satisfying equation 9 required about a 4% sample; for a $131K$ query TPC-D workload, a samples of less than 0.6% of the queries was needed.

7 Experiments

In this section we first evaluate the efficiency of the sampling techniques described in Sections 4 and 5. Then, we show how the estimation of $Pr(CS)$ can be used to construct a scalable and accurate comparison primitive for large numbers of configurations. Finally, we compare our approach to existing work experimentally.

Setup: All experiments were run on a Pentium 4 (2.8GHz) PC and use the cost model of the SQL Server optimizer. We evaluated our techniques on two databases, one synthetic, and one real-life. The synthetic database follows the *TPC-D* schema and was generated so that the frequency of attribute values follows a Zipf-like distribution, using the skew-parameter $\theta = 1$. The total data size is $\sim$1GB. Here, we use a workload consisting of about 13K queries, generated using the standard QGEN tool. The real-life database used was a database running a CRM application with over 500 tables and of size $\sim$0.7 GB. We obtained the workload for this database by using a trace tool; the resulting workload contains about 6K queries, inserts, updates and deletes.

7.1 Evaluating the Sampling Techniques

In this section, we evaluate the efficiency of the different sampling techniques described (Independent and Delta Sampling, both with and without progressive stratification). In the first experiment, we use the *TPC-D* workload and consider the problem of choosing between two configurations C_1, C_2 that have a significant difference in cost (7%) and in their respective sets of physical design structures (C_2 is index-only, whereas C_1 contains a number of views). Note that $|\mathcal{WL}| = \sim$13K, so solving the configuration-selection problem for $k = 2$ exactly requires $\sim$26K optimizer calls. In the first experiment we set $\delta = 0$ and run each sampling scheme for a given sample size and output the selected configuration. This process is repeated 5000 times, resulting in a Monte Carlo simulation to compute the 'true' probability of correct selection for a given number of samples. The results are shown in Figure 1. We can see that the sampling-based approach to configuration selection is efficient: less than 1% of the number of optimizer calls required to compute the configuration costs exactly suffice to select the correct configuration with near-certainty. Delta-Sampling outperforms Independent Sampling significantly for small sample sizes, whereas adding progressive stratification makes little difference, given the small sample sizes.

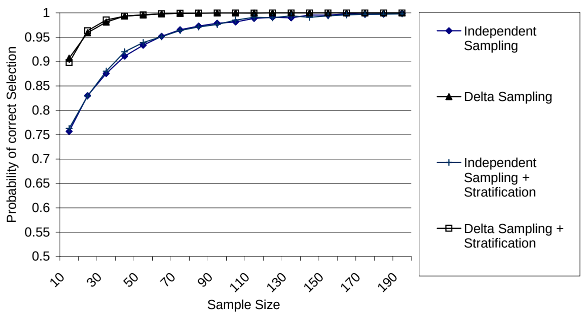

Figure 1: Monte Carlo Simulation of $Pr(CS)$

In the next experiment we illustrates the issues of initially picking a very fine stratification of the workload. Here, we run the same experiment again, and use both sampling schemes when having partitioned the workload into L strata, one for each query template (Figure 2). For the fine stratification and small sample sizes, the estimates within each stratum are not normal and thus the probability of correct selection is significantly lower. For large sample sizes, the accuracy of the fine stratification is comparable to the progressive stratification scheme. We have found this to hold true for a number of different experimental setups.

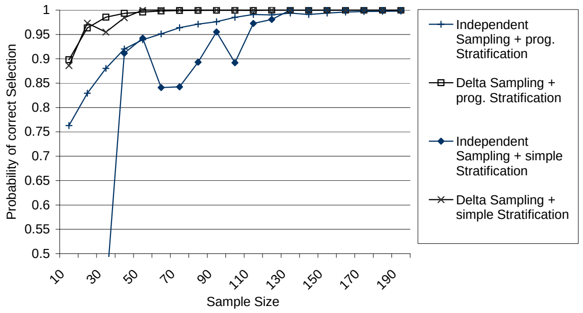

Figure 2: Progressive vs. fine stratification

The next experiment (Figure 3) uses the same workload, but two configurations that are significantly harder to distinguish (difference in cost $\leq 2\%$), and share a significant number of design structures (both configurations are index-only). Here, Delta Sampling outperforms Independent Sampling by a bigger margin, since the configurations share a large number of objects, resulting in higher covariance between the cost distributions. Because of the larger sample sizes, stratification significantly improves the accuracy of Independent Sampling.

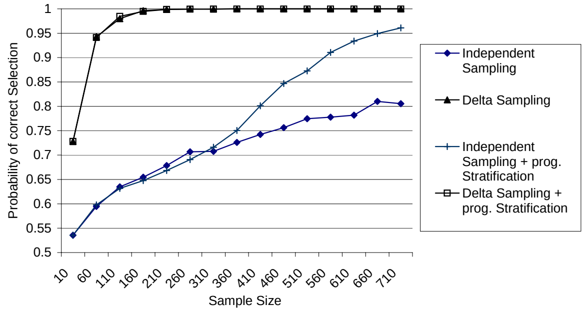

Figure 3: Monte-Carlo Simulation of $Pr(CS)$

We conducted the same experiments on the real-life database, using two configurations that were difficult to compare (difference in cost $< 1\%$), and had little overlap in the physical design structures (Figure 4). Consequently, the advantage of Delta Sampling is less pronounced. Furthermore, this workload contains a relatively large number of distinct templates (> 120). This means that we rarely have estimates of the avg. cost of *all* templates, so progressive stratification is used only in a few cases.

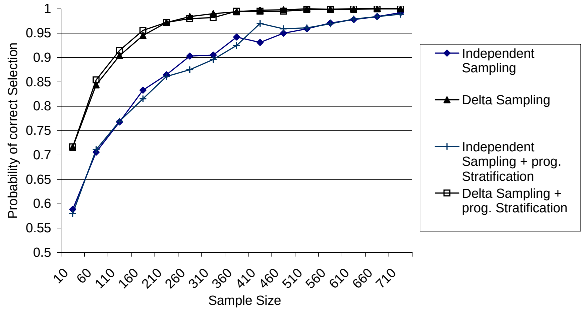

Figure 4: Monte-Carlo Simulation of $Pr(CS)$

7.2 Experiments on multiple configurations

In this section we evaluate the accuracy and scalability of the comparison primitive based on $Pr(CS)$-estimates for large numbers of configurations k, which were collected from a commercial physical design tool. We run Algorithm 1 to determine the best configuration with probability $\alpha = 90\%$, $\sigma = 0$ and using Delta-Sampling with progressive stratification as the sampling technique.

In these experiments, we use the sample variances ${s_i}^2$ to estimate ${\sigma_i}^2$. In order to guard against the oscillation of the $Pr(CS)$-estimates, we only accept a $Pr(CS)$-condition if it holds for more than 10 consecutive samples. As an additional optimization, we stop sampling from configurations C_j for which the contribution to the overall uncertainty in $Pr(CS)$ is negligible (in this case $Pr(CS_{l,j}) > .995$).

We compare the primitive to two alternative sample-allocation methods (using identical number of samples): (a) sampling without stratification and (b) sampling the same number of queries from each stratum. For all methods, we report the resulting probability of selecting the best configuration ("True $Pr(CS)$") and the maximum difference in cost between the best configuration and the one selected by each method ("Max. Δ"). The latter allows us to assess the worst-case impact of using the alternative techniques. Each experiment was carried out 5000 times, for both the TPC-D (Table 2) and the CRM database (Table 3).

Method		$k = 50$	$k = 100$	$k = 500$
Delta-Sampling	True $Pr(CS)$	91.7%	88.2%	88.3%
	Max. Δ	0.5%	1.5%	1.6%
No Strat.	True $Pr(CS)$	39.1%	28.2%	12.0%
	Max. Δ	8.8%	9.9%	9.8%
Equal Alloc.	True $Pr(CS)$	42.5%	28.6%	12.8%
	Max. Δ	7.7%	9.0%	8.6%

Table 2: Results for TPCD-Workload

Method		$k = 50$	$k = 100$	$k = 500$
Delta-Sampling	True $Pr(CS)$	97.5%	94.4%	89.7%
	Max. Δ	1.7%	1.4%	0.8%
No Strat.	True $Pr(CS)$	56.0%	37.5%	11.0%
	Max. Δ	10.53%	12.69%	6.5%
Equal Alloc.	True $Pr(CS)$	71.1%	52.8%	17.0%
	Max. Δ	7.2%	5.8%	3.26%

Table 3: Results for CRM-Workload

In both cases, the comparison primitive based on Delta-Sampling performs significantly better than the alternatives. Moreover, the true $Pr(CS)$ resulting from the selection primitive matches the target probability α closely or exceeds it[4], demonstrating its capability to chose the sample size effectively to match the accuracy requirement.

7.3 Comparison to Workload Compression

In this section, we compare existing workload compression technqiues [5, 20] to our approach with three key evaluation criteria: *scalability*, *quality* and *adaptivity*. In [20], queries are selected in order of their costs for the current configuration until a prespecified percentage, X, of the total workload cost is selected. Obviously, this method scales well. Regarding quality, however, the technique fails when only few query templates contain the most expensive queries. To illustrate this, we generated a 2K query TPC-D workload using the QGEN tool. If $X = 20\%$, [20] will capture queries corresponding to only few of the TPC-D query

[4] The high $Pr(CS)$ for the CRM workload is due to the requirement that $Pr(CS) > \alpha$ must hold for 10 consecutive samples, which causes over-sampling for easy selection problems.

templates. Consequently, tuning this compressed workload fails to yield several design structures beneficial for the remaining templates. To show this, we tuned 5 different random samples of the same size as the compressed workload; the improvement (over the entire workload) resulting from tuning each sample was more than twice the improvement resulting from tuning the compressed workload.

To evaluate the quality resulting from [5], we measured difference in improvement when tuning a Delta-sample and a compressed workload of the same size for a TPC-D workload; in this experiment, both approaches performed comparably. Regarding scalability, [5] requires up to $O(|\mathcal{WL}|^2)$ complex distance-computations as a preprocessing step to the clustering problem. In contrast, the overhead for executing the Algorithms 1 and 2 is negligible, as all necessary counters and measurements can be maintained incrementally at constant cost.

The biggest difference between [5, 20] and our technique concerns adaptivity. [5, 20] both depend on an initial guess of a sensitivity parameter. In [5], it is the maximum allowable increase in the estimated running time when queries are discarded and in [20] it is the percentage of total cost retained. It is not clear how to set this parameter correctly, as these deterministic techniques do not allow the formulation of probabilistic guarantees similar to the ones in this paper; in both [5, 20] the sensitivity parameter is set up-front, without taking the configurations space into account. We have observed in experiments that the fraction of a workload required for accurate selection varies significantly for different sets of candidate configurations. Thus choosing the sensitivity parameter incorrectly has significant impact on tuning quality and speed. Our algorithm, in contrast, offers a principled way of adjusting the sample size online.

8 Conclusion and Outlook

In this paper, we presented a scalable comparison primitive solving the *configuration selection problem*, while providing accurate estimates of the probability of correct selection. This primitive is of crucial importance to scalable exploration of the space of physical database designs and can serve as a core component within automated physical design tools. Moreover, we described a novel scheme to verify the validity of using the Central Limit Theorem and sample variances to compute $Pr(CS)$. These techniques are not limited to this problem domain, but can be used for any CLT-based estimator, if the required bounds on individual values in the underlying distribution can be obtained.

Acknowledgements: This paper benefitted tremendously from many insightful comments from Vivek Narasayya, Surajit Chaudhuri, Sanjay Agrawal and Amin Saberi.

References

[1] S. Agrawal, S. Chaudhuri, et al. Database Tuning Advisor for Microsoft SQL Server. *Proc. of the 30th VLDB Conf., Toronto, Canada*, 2004.

[2] N. Bruno and S. Chaudhuri. Automatic Physical Database Tuning: a Relaxation-based Approach. In *ACM SIGMOD Conf.*, 2005.

[3] G. Casella and R. L. Berger. *Statistical Interference*. Duxbury, 2002.

[4] S. Chaudhuri, G. Das, et al. A Robust, Optimization-Based Approach for Aprroximate Answering of Aggregate Queries. In *Proc. of ACM SIGMOD Conf.*, 2001.

[5] S. Chaudhuri, A. Gupta, et al. Compressing SQL Workloads. In *Proc. of ACM SIGMOD Conf.*, 2002.

[6] S. Chaudhuri, A. C. König, et al. SQLCM: A Continuous Monitoring Framework for Relational Database Engines. In *Proc. of 20th I CDE Conf.*, 2004.

[7] S. Chaudhuri and V. R. Narasayya. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proc. the 23rd VLDB Conf., Athens, Greece*, 1997.

[8] S. Chaudhuri and V. R. Narasayya. AutoAdmin ”What-If” Index Analysis Utility. In *Proc. of ACM SIGMOD Conf., Seattle, WA, USA*, 1998.

[9] W. G. Cochran. *Sampling Techniques*. Wiley, 1977.

[10] B. Dageville, D. Das, et al. Automatic SQL Tuning in Oracle 10g. In *Proc. of the 30th VLDB Conf.*, 2004.

[11] S. Ferson, L. Ginzburg, et al. Computing Variance for Interval Data is NP-Hard. In *ACM SIGACT News, Vol. 33, No. 2*, pages 108–118, 2002.

[12] S. Ferson, L. Ginzburg, et al. Exact Bounds on Finite Populations of Interval Data. Technical Report UTEP-CS-02-13d, University of Texas at El Paso, 2002.

[13] S. Finkelstein, M. Schikolnick, et al. Physical Database Design for Relational Databases. In *ACM TODS, 13(1)*, 1988.

[14] Y. Ioannidis and S. Christodoulakis. Optimal Histograms for limiting Worst-Case Error Propagation in the Size of Join Results. In *ACM TODS*, 1993.

[15] S.-H. Kim and B. L. Nelson. Selecting the Best System: Theory and Methods. In *Proc. of the 2003 Winter Simulation Conf.*, pages 101–112.

[16] V. Kreinovich, S. Ferson, et al. Computing Higher Central Moments for Interval Data. Technical Report UTEP-CS-03-14, University of Texas at El Paso, 2003.

[17] N. M. Steiger and J. R. Wilson. Improved Batching for Confidence Interval Construction in Steady-State Simulation. In *Proc. of the 1999 Winter Simulation Conf.*, 2000.

[18] M. Stillger, G. Lohman, V. Markl, and M. Kandil. LEO - DB2’s Learning Optimizer. In *Proc. of the 27th Conference on Very Large Databases*, 2001.

[19] R. Sugden, T. Smith, et al. Cochran’s rule for simple random sampling. In *Journal of the Royal Statistical Society*, pages 787–793, 2000.

[20] D. C. Zilio, J. Rao, et al. DB2 Design Advisor: Integrated Automatic Physical Database Design. *Proc. of the 30th VLDB Conf., Canada*, 2004.