Random Sampling from Pseudo-Ranked B^+ Trees

Gennady Antoshenkov
Data Base Systems Group, Digital Equipment Corporation
55 Northeastern Blvd., Nashua, NH 03062, USA
antoshenkov@nova.enet.dec.com

Abstract

In the past, two basic approaches for sampling from B+ trees have been suggested: sampling from the ranked trees and acceptance/rejection sampling from non-ranked trees. The first approach requires the entire root-to-leaf path to be updated with each insertion and deletion. The second has no update overhead, but incurs a high rejection rate for the compressed-key B+ trees commonly used in practice. In this paper we introduce a new sampling method based on pseudo-ranked B+ trees, which are B+ trees supplemented with information loosely describing the estimated rank limits. This new method exhibits a very small rejection rate while paying only a marginal cost of the tree update overhead. We also present comparative efficiency measurements of different methods that were run on production databases and on several prototype workload simulations.

1. Introduction

A number of database applications, like financial auditing [Ark84] or quality control [LWW84], rely on extraction of a small portion of records at random from a large data collection. Without a good sampling mechanism, such random sample extraction may be expensive when running on traditional DBMS. One should perform a full scan of the most compact index, collect record IDs, do random sampling from the record ID list, and then fetch the selected records. An alternative way of sampling is to pick an index with the flattest B+ tree, repeatedly descend the tree in random directions, reject some of the attempts to achieve the unbiased randomness (if needed), and fetch the records for the successful descents. The latter approach supersedes the previous one and constitutes a foundation of a majority of sampling methods that use B+ trees.

Proceedings of the 18th VLDB Conference
Vancouver, British Columbia, Canada 1992

A demand for *efficient* sampling techniques is even higher when random samples are used for estimation of different quantities involved in aggregate, join, or select operations. In recent publications, random samples serve for building approximate selectivity histograms [PiCo84,MuDe88], for estimation of aggregate queries [HOT88,HOT89], for estimation of query size [LiNa89,LiNa90,LNS90]. R. Lipton, J. Naughton, and D. Schneider in [LNS90] presented new low sample size bounds for estimation with a given precision and confidence thus proving sample-based estimation is a more practical technique than was previously thought.

However, sample-based estimation can only be as efficient as the method of obtaining the unbiased random samples. To see the realistic performance figures of the major existing sampling methods, we analyzed the B+ trees of all the indexes in one of our customer databases using DEC's Rdb/VMS * relational runtime. The results were discouraging. The biggest ($\approx$900K) index would require more than three thousand attempts to descend its B+ tree before one attempt is accepted as valid for simple random sampling using the acceptance/rejection (A/R) algorithm described in [OlRo89]. In section 3, we discuss this experiment in more detail, construct a really bad "worst case" index, and offer explanations for a poor performance of A/R sampling on large B+ trees used in commercial databases.

A three decimal order rejection rate, as compared to the theoretically fastest single B+ tree descent per sample, makes random sampling a bottleneck for sample-based estimation algorithms.

In this paper we present a new method of sampling from B+ trees supplemented with rank-related information. With this method:

(1) Sample acceptance rate is on the order of 50%.

* DEC and Rdb/VMS are trademarks of Digital Equipment Corporation.

(2) Acceptance rate stays stable for a variety of the tree sizes and degrees of index key compression.
(3) Tree maintenance overhead is on the order of 1%.
(4) Sampling performance can be balanced with maintenance overhead as well as tuned to different tree shapes.

The new method, called random sampling from pseudo-ranked B+ trees, unifies the two known basic approaches: sampling from ranked trees [Knu73,SrLu88] and A/R sampling from non-ranked trees [OlRo89]. Remarkably, this unification does not just break a middle ground between the two extremes and achieve tunability, but rather takes the best from both worlds, minimizing both overhead and sampling time.

There are a few important variations of the basic sampling algorithms: sampling with and without replacement, iterative and batch sampling. They are all applicable to "pseudo-ranked" sampling, but we will not discuss them here. Instead we refer to their detailed description in F. Olken and D. Rotem [OlRo89].

The rest of this paper is organized as follows. Overviews of "ranked" and "A/R" sampling techniques are given in sections 2 and 3 respectively. A new method of simple random sampling from pseudo-ranked B+ trees is introduced in section 4. Section 5 describes a calculation scheme for pseudo-ranked tree maintenance, including a variant which simulates A/R sampling. Section 6 contains the performance evaluation results. A summary and directions for future work are given in section 7.

2. Sampling from Ranked Trees

Simple random sampling from a table is a selection of one or more of its records with equal probability across all records and with no dependencies between different trials. *Sampling from B+ tree* is sampling of records from its underlying table by means of traversing a B+ tree structure. Randomness of selections implies a total independence of a sampling method from the B+ tree keys, separators, and node fanouts. When descending a B+ tree with random sampling in mind, all key and separator values can be simply ignored. But different densities of node fanouts in different tree areas may significantly distort a tree shape uniformity and thus require special mechanisms to correct such distortions.

We will further concentrate only on simple random sampling from B+ trees and explore several mechanisms for distorted tree shape compensation. Also, to simplify algorithms, we will consider only B+ trees with two or more levels, because any sampling from a single-level tree is a trivial matter.

Suppose that each record in a table corresponds to one key in a B+ tree and all keys are ranked from 1 to K following the tree sequence. A B+ tree can be supplemented with information which allows one to find a k-th ranked key with one tree descent. Such trees are called *ranked* trees. Based on ranked B+ trees one can draw a random sample by first picking an equi-probable $k \in [1, K]$, then descending to the k-th key and fetching the record. This idea of a direct or random k-th key access via a ranked B+ tree is discussed in [Knu73,BeKr75,SrLu88]

Notations. Let N be an arbitrary B+ tree node having f children $n_1, ..., n_f$ or containing f keys (leaf node case), where f is a fanout of node N. Let c_i be a cardinality of child n_i (number of keys in n_i's branch) and $r_i = \sum_{j \le i} c_j$ be a rank of child n_i (rank of n_i's last key) within its parent's branch. When needed, we will also use $f(N)$, $n_i(N)$, $c_i(N)$, $r_i(N)$ notations to directly specify a desired node and avoid ambiguity. Finally, let T be a tree top (root) node so that the tree cardinality $K = r_f(T)$, and let's use $r_0 = 0$ notation for the convenience of the reverse calculation of cardinalities: $c_i = r_i - r_{i-1}$.

If each non-leaf node is supplemented with cardinalities or ranks of its children stored along with the pointers to children, then such B+ tree is a ranked B+ tree with a "descend to k-th key" procedure defined by the following algorithm.

1. Start from top: $N \leftarrow T$
2. Pick arbitrary $k \in [1, K]$
3. Find i: $r_{i-1} < k \le r_i$
4. Reset k: $k \leftarrow k - r_{i-1}$
5. Descend: $N \leftarrow n_i$
6. If N is not leaf, go to 3
7. Deliver record k from N

Algorithm 2: Sampling from Ranked Tree

Sampling from the ranked B+ trees is the fastest possible, one-descent procedure. It avoids a tree shape uniformity distortion by providing a direct access to the k-th key. The direct access power of ranked B+ trees broadens their usage to include such applications as a quick location of the k-th byte in large continuous objects [CDRS86]. At the same time, maintenance of ranked B+ trees incurs a top cost of updating ranks along the entire root-to-leaf path for each insert/delete operation. This high maintenance cost, along with a necessity of frequent root node locking, limits any usage of ranked B+ trees, including sampling, only to the low update rate environments.

3. A/R Sampling from Non-Ranked Trees

A very different sampling method, described in this section, is based on exercising a random choice at each level of descent by picking a random child to descend. In this method, a non-uniform distribution of fanouts is compensated by extending a given B+ tree to an equi-fan tree, descending it, and applying an acceptance/rejection technique to pick only the nodes of the original tree.

Let $F_{max} = max_N f(N)$ be a maximum fanout and $F \geq F_{max}$ be an upper fanout bound effectively guaranteed for a given tree. Figure 3 depicts the original 3-level B+ tree and its equi-fan extension. Note that the extended tree never materializes, but serves merely for conceptual clarity.

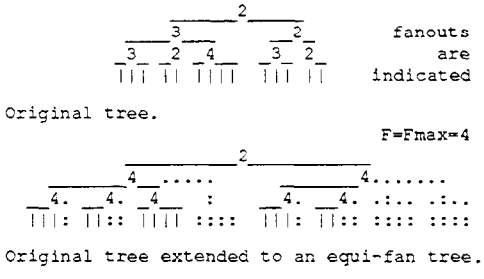

Figure 3: Example of Equi-Fan Tree Extension

The following A/R algorithm describes a simple random sampling procedure presented in [OlRo89] as an "early abort method". (Application of A/R techniques to sampling from relational operators can be found in [OlRo86].)

1. Pick random root child $N \leftarrow n_i(T)$
2. Pick random $k \in [1 : F]$
3. If $k > f$, reject: go to 1
4. If N is leaf, conclude: go to 7
5. Descend: $N \leftarrow n_k$
6. Go to 2
7. Deliver the k-th record from N

Algorithm 3: A/R Sampling from Non-Ranked Tree

Unlike sampling from ranked trees, the A/R method uses only fanouts and child pointers available in any traditional B+ tree structures. Therefore, it requires no extra effort to maintain rank-related information. The price for providing an equi-probable selection of records is paid when some descent attempts are rejected.

The descent reject rate largely depends on the height of the tree and F_{max}/F_{avg} proportion where F_{avg} is the average fanout across all non-root nodes. In turn, the F_{max}/F_{avg} ratio largely depends on the method of B+ tree balancing. As described in [Knu73], node split, merge, or rebalance can be triggered either when a fanout gets out of its predetermined limits or when the node page space overflows /underflows (suggested by T.H. Martin for variable length keys.) Most commercial databases today use various key compression techniques and thus manage rebalancing by node page space.

To obtain realistic performance figures for A/R sampling, we analyzed all indexes in a production database whose B+ trees are space-balanced. We also obtained analogous performance figures for fan-balanced B+ trees by simulating a "random insert" workload and then analyzing the created trees.

	Space-balanced B+ trees		
Rejection rate *	223.2	725.3	3,376.5
F_{max}/F_{avg}	2.1	2.7	3.4
Tree height	4	5	6
	Fan-balanced B+ trees, $f \in [5..10]$		
Rejection rate *	2.90	4.17	6.15
F_{max}/F_{avg}	1.341	1.326	1.322
Tree height	4	5	6

* Rejection rate is the average number of rejects per single sample acceptance.

Table 3: A/R Sampling Performance

The results for fan-balanced trees in table 3 are close to the approximate performance curve presented in figure 1 in [OlRo89]. However, the rejection rates for realistic space-balanced trees are sharply higher, expressed in hundreds and thousands of rejects per single acceptance. For a 6-level tree this translates into an I/O cost that is several hundred times higher than the cost of sampling from ranked trees. Another negative performance factor should be added on top of the demonstrated rejection rate, namely the F/F_{max} ratio. The upper bound F, when deviating from F_{max}, has a strong negative performance impact. And the task of maintaining F close to F_{max} is difficult and expensive for space-balanced trees when a significant "delete" rate is anticipated.

Finally, we tested the rejection rate stability for B+ tree key patterns that lead to low fanout uniformity. We created an index with a 3-attribute composite key
$\langle integer, char(100), integer \rangle$,
and loaded 8192 values
$\langle i/2$, "100-char-constant", $i \rangle$, $i \in [1, 8192]$.
The results of this B+ tree analysis, even assuming $F = F_{max}$, were astonishing:

Rejection rate	77,396,705,279	
F_{max}/F_{avg}	6	(=24/4)
Tree height	10	levels

77 billion rejects per one acceptance on less than 10K tree indicates a high instability of A/R sampling when done on space-balanced B+ trees.

4. Sampling from Pseudo-Ranked Trees

One can observe that the two major sampling methods described above reveal their power and weaknesses on sampling cost / update overhead scale in the opposite way. A new sampling method presented below cuts the weaknesses and retains the strengths of "ranked" and "A/R" approaches by making their different descent principles work together.

Extending section 2 definitions, let $c(n)$ be a cardinality of node n, i.e. a number of keys in n's branch. For non-leaf node N, let's define some upper bounds c_i^+ of N's children cardinalities and store them in N along with the pointers to children n_i. We also want the c_i^+ bounds be the bounds of all lower-level bounds in n_i's subtree. This "nested bound" property can be enforced by imposing a *nested bound* condition

$c_i^+(N) \geq \sum_{j=1}^{f(n_i)} c_j^+(n_i)$ or

$c_i^+(N) \geq c(n_i)$ when n_i's are leaves

for all parent/child pairs N, n_i. Symmetrically, we define lower bounds c_i^- of N's children cardinalities, store them in N, and impose the nested bound condition

$c_i^-(N) \leq \sum_{j=1}^{f(n_i)} c_j^-(n_i)$ or

$c_i^-(N) \leq c(n_i)$ when n_i's are leaves.

And further, we define upper and lower rank bounds as

$r_i^+ = \sum_{j \leq i} c_j^+,\ r_0^+ = 0$ and

$r_i^- = \sum_{j \leq i} c_j^-,\ r_0^- = 0$.

Either cardinality or rank bounds can be stored in parent nodes, whichever is more efficient for a given implementation. Rank bounds make possible a binary search within a node, but require a multiple correction, namely corrections of $r_i^+, ..., r_f^+$, $r_i^-, ..., r_f^-$ during update.

It is not necessary to store both upper and lower cardinality (or rank) bounds. One can store a single quantity c_i^o for each child and calculate $c_i^+ = Upper(c_i^o)$, $c_i^- = Lower(c_i^o)$ upon need using *Upper* and *Lower* functions defined for a given tree or system-wide.

Definition. Regardless of a storing method, we will call a *pseudo-ranked* tree any B+ tree supplemented with information that allows calculation of upper and lower cardinalities/ranks satisfying the nested bound conditions for all parent/child pairs.

Maintenance of pseudo-ranked trees is a straightforward procedure. Upon insert or delete operation, the nested bound conditions are evaluated for each parent and child involved in node's split, merge, and rebalancing. Once the condition breaks, new bounds, satisfying the nested bound condition, are calculated and stored in a parent node. If a parent node is updated due to reorganization of its children, the fresh bounds are calculated and stored in the parent to improve a precision even in case of the nested bound condition satisfaction. Tree reorganization proceeds up toward the root, leaving below only "correct" rank bounds. The bound condition check and correction proceeds toward the root even after a local tree reorganization is accomplished and stops when the nested bound condition satisfies.

To define a sampling procedure for pseudo-ranked trees, we first describe an extension of an arbitrary pseudo-ranked tree to some ranked B+ tree. Let

$\Delta_i(N) = c_i^+(N) - \sum_{j=1}^{f(n_i)} c_j^+(n_i)$ or

$\Delta_i(N) = c_i^+(N) - c(n_i)$ if n_i's are leaves

be the bound-related cardinality excesses, all non-negative due to the nested bound condition. For each non-leaf child n_i of parent N with $\Delta_i(N) > 0$ we add an extra grand child and a single-thread branch of its grand...grand children with a leaf-level grand...grand child having $\Delta_i(N)$ dummy keys. For each leaf child n_i we simply add $\Delta_i(N)$ dummy keys to n_i. If we set $c_i^+(N) = c(n_i)$ for all new nodes n_i, then c_i^+ bounds become real cardinalities through the entire extended tree, making it a (fully) ranked tree.

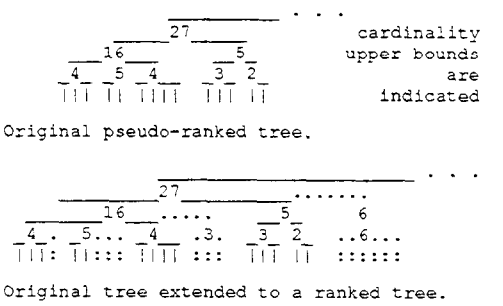

Figure 4: Pseudo-Ranked Tree Extension

Sampling from a pseudo-ranked tree can now be defined as a sequence of descents into an extended ranked tree with a descent attempt rejected when an extra, non-original child or dummy key is picked. Algorithm 4 describes this sampling procedure, using the extended ranked tree only as a conceptual model without ever materializing it.

1. Start from top: $N \leftarrow T$
2. Pick random $k \in [1, r_f^+]$
3. If $k > r_f^+$, reject: go to 1
4. Find i: $r_{i-1}^+ < k \leq r_i^+$
5. Reset k: $k \leftarrow k - r_{i-1}^+$
6. Descend: $N \leftarrow n_i$
7. If N is not leaf, go to 3
8. If $k > c(N)$, reject: go to 1
9. Deliver record k from N

Algorithm 4: Sampling from Pseudo-Ranked Tree

Applied to a ranked extension of the original pseudo-ranked tree, algorithm 4 performs no rejections and acts exactly as "ranked sampling" algorithm 2, selecting each key $k \in [1, r_f^+(T)]$ with equal probability $1/r_f^+(T)$. The extended tree descents, that pick keys (and records) from the original tree, coincide precisely with records selected by algorithm 4 applied to the original tree. Since all descent attempts are independent, selection of records from the original subtree of the extended tree is also equiprobable, no matter to which of the two trees algorithm 4 is applied. This proves the following theorem.

Theorem 4 *Sampling from a pseudo-ranked B+ tree by one or multiple applications of algorithm 4 is a simple random sampling from the underlying table. Probability of selecting each record is* $1/c(T)$.

From the extended tree definition it follows that the extended tree cardinality equals the rank bound $r_f^+(T)$ of the last child of the root of the original tree. Further,

$r_f^+(T) = c(T) + \sum_{(N.n_i)} \Delta_i(N)$,

where $c(T)$ is the original tree cardinality. The algorithm 4 rejection rate can now be expressed as

$(r_f^+(T) - c(T))/c(T) = \sum_{(N.n_i)} \Delta_i(N)/c(T)$.

This means that when setting new upper bounds during insert/delete operations, the excess quantity $\Delta_i(N)$, relative to the lower level bounds, contributes directly to the rejection rate increase. On the other hand, $\Delta_i(N)$ determines the margins for maintenance-free inserts/deletes performed within n_i's branch and thus contributes positively to the maintenance overhead reduction.

If we only do inserts, only the upper rank bounds can be violated. In this case, maintenance of upper bounds only is sufficient. When we do multiple deletes and do not maintain lower bounds, the upper bounds will remain the same causing a constant growth of rejection rate $(r_f^+(T)/c(T) - 1)$ due to a tree cardinality decrease. Therefore, maintenance of the lower bounds r_i^- is necessary for controlling the rejection rate when deletes are considered.

5. Bound Maintenance

There are many ways of setting new cardinality or rank bounds when the nested bound condition breaks. We will successively narrow this task's generality and come up with a calculation scheme that gives good sampling performance. We will also describe a simulation of A/R method, and incorporate its advantages into sampling from pseudo-ranked trees.

Step 1. For any child n_i, let's store only one, cardinality-related quantity c_i^o in n_i's parent and calculate

$c_i^+ = c_i^o * (1+\epsilon)$, $c_i^- = c_i^o/(1+\epsilon)$,

$r_i^+ = \sum_{j \le i} c_j^+$, $r_i^- = \sum_{j \le i} c_j^-$ upon need, assuming ϵ is a function of any information related to a given B+ tree and $\epsilon \ge 0$.

Step 2. Assume that in different areas of a single tree level a tree shape development has insignificant bearing on rank bounds, or that it pays to ignore the horizontal shape non-uniformity. Then we can further assume that for a given tree, ϵ depends only on the tree level of child n_i. If $h(n)$ is a height of a node n branch, a finite set $\{\epsilon_h\}_{h \in [1..h(T)]}$ may be used in bounds calculation, like

$c_i^+ = c_i^o * (1 + \epsilon_{h(n_i)})$ or simply

$c_i^+ = c_i^o * (1 + \epsilon_h)$ since the node is known.

Step 3. When the bound recalculation is needed for parent/child pair N, n_i, let's set

$c_i^o(N) = \sum_{j=1}^{f(n_i)} c_j^o(n_i)$ if $h(n_i) > 1$ and

$c_i^o(N) = c(n_i)$ if $h(n_i) = 1$.

Let's also consider only non-decreasing $\epsilon_1, ..., \epsilon_{h(T)}$ sequences, $\epsilon_h \ge \epsilon_{h-1}$ for any $h > 1$, because they guaranty satisfaction of the nested bound condition.

Indeed, if $h = h(n_i)$ and $h > 1$, then

$\sum_{j=1}^{f(n_i)} c_j^+(n_i) = \sum_{j=1}^{f(n_i)} (c_j^o(n_i)(1+\epsilon_{h-1})) =$

$(1+\epsilon_{h-1}) \sum_{j=1}^{f(n_i)} c_j^o(n_i) = (1+\epsilon_{h-1}) c_i^o(N)$,

$c_i^o(N) = \frac{1}{1+\epsilon_{h-1}} \sum_{j=1}^{f(n_i)} c_j^+(n_j)$, and

$c_i^+(N) = c_i^o(N)(1+\epsilon_h) = \frac{1+\epsilon_h}{1+\epsilon_{h-1}} \sum_{j=1}^{f(n_i)} c_j^+(n_j)$.

The following also holds:

$c_i^-(N) = \frac{1+\epsilon_{h-1}}{1+\epsilon_h} \sum_{j=1}^{f(n_i)} c_j^-(n_j)$ if $h = h(n_i) > 1$,

$c_i^+(N) = (1+\epsilon_1) c(n_i)$ if $h(n_i) = 1$,

$c_i^-(N) = \frac{1}{1+\epsilon_1} c(n_i)$ if $h(n_i) = 1$.

Now we see that the nested bound condition holds for newly recalculated bounds because $\epsilon_h \ge \epsilon_{h-1} \ge 0$ and ϵ-dependent terms are ≥ 1 for upper bounds and are ≤ 1 for lower bounds.

Step 4. Let's limit $\epsilon_1, ..., \epsilon_{h(T)}$ sequences to sequences $\epsilon_h = \prod_{i=0}^{h-1}(1 + A * Q^i) - 1$ based on geometric sequences $A * Q^{h-1}$ with a first term A and a common ratio Q, $A \ge 0$, $Q \ge 0$.

A/R simulation. Let F be some upper fanout bound and $c_i^+ = F^{h(n_i)}$, $c_i^- = 0$ be cardinality bounds for all parent/child pairs N, n_i. A probability of acceptance of picking a grandchild or a key of child n_i when sampling from such pseudo-ranked B+ tree is

$p_{acc}(n_i) = \frac{\sum_{j=1}^{f(n_i)} c_j^+(n_i)}{c_i^+(N)} = \frac{\sum_{j=1}^{f(n_i)} F^{h(n_i)-1}}{F^{h(n_i)}} = \frac{f(n_i)}{F}$ or

$p_{acc}(n_i) = \frac{c(n_i)}{c_i^+(N)} = \frac{f(n_i)}{F}$ if $h(n_i) = 1$.

This matches exactly an acceptance probability of A/R sampling from non-ranked tree at all (non-root) nodes n_i. To simulate a "non-ranked" sampling driven by $k_1, ..., k_m, ...$ random sequence with $k_m \in [1, F]$, one can compose $k = \sum_{m=1}^{h(T)} ((k_m - 1) F^{h(T)-m}) + 1$ as a random number from $[1, F^{h(T)}]$ and apply it at step 2 of algorithm 4. After rejection or acceptance, repeat the k composition starting with a k_μ next after the last one used in a parallel run of algorithm 3.

Step 5. For any calculation scheme S delivering c_i^+ and c_i^- cardinality bounds and for any upper fanout bound F, let's use a new scheme $\widehat{S}$ with bounds $\hat{c}_i^+ = min(c_i^+, F^{h(n_i)})$, $\hat{c}_i^- = c_i^-$ for each parent/child pair N, n_i.

New bounds satisfy the nested bound condition because
$\sum_{j=1}^{f(n_i)} \hat{c}_j^+(n_i) = \sum_{j=1}^{f(n_i)} min(c_j^+(n_i), F^{h-1}) \le$
$min(\sum_{j=1}^{f(n_i)} c_j^+(n_i), \sum_{j=1}^{f(n_i)} F^{h-1}) \le$
$min(c_i^+(N), F^h) = \hat{c}_i^+(N)$ if $h = h(n_i) > 1$,
$c(n_i) \le c_i^+(N)$ and $c(n_i) \le F$ yield
$c(n_i) \le min(c_i^+(N), F) = \hat{c}_i^+(N)$ if $h(n_i) = 1$,
and because the lower bounds remain unchanged.

Definition. For a given ranked, pseudo-ranked, or non-ranked tree, let C be the average cost of obtaining a single sample from this tree measured in units of processed nodes, excluding the root node T which is assumed to be a memory resident, hence making no contribution to the node fetch I/Os.

Lemma 5 *Let $\hat{c}_i^+$ and c_i^+ be upper bounds of two pseudo-ranked trees based on a common B+ tree, $\hat{c}_i^+ \le c_i^+$ for all parent/child pairs N, n_i. If we apply algorithm 4 for sampling from both trees, the rejection rate (point one) and cost C (point two) of sampling from the first tree do not exceed those from the second tree.*

Proof: Using upper rank bounds, rejection rates are expressed as
$\hat{R} = \hat{r}_f^+(T)/c(T) - 1 = \sum_{i=1}^{f(T)} \hat{c}_i^+(T)/c(T) - 1$,
$R = r_f^+(T)/c(T) - 1 = \sum_{i=1}^{f(T)} c_i^+(T)/c(T) - 1$,
Combined with $\hat{c}_i^+ \le c_i^+$, it yields $\hat{R} \le R$, and thus proves the first point.

For bounds c_i^+, let's consider all different equiprobable descent attempts starting with $\bar{k} = 1, ..., r_f^+(T)$ as selected in step 2 of algorithm 4. It is easy to verify that for each parent/child pair N, n_i, any k set by step 5, $k \in [1, c_i^+(N)]$, has exactly one $\bar{k} \in [1, r_f^+(T)]$ which determines this particular descent to child n_i. Therefore, the descent to child n_i in step 6 is performed exactly $c_i^+(N)$ times in all descent attempts with $\bar{k} = 1, ..., r_f^+(T)$.

Now note that each single-level descent in step 6 accounts for one cost unit and that there are no other steps in algorithm 4 which contribute to the cost increase. The cached root node fetch in step 1 does not count. We can now express the total cost of all descent attempts as $C_{total} = \sum_{(N.n_i)} c_i^+(N)$ and the average cost to obtain a single sample as $C = \sum_{(N.n_i)} c_i^+(N)/c(T)$.

Comparing $\widehat{C} = \sum_{(N.n_i)} \hat{c}_i^+(N)/c(T)$ with C under assumption $\hat{c}_i^+ \le c_i^+$, we come up with $\widehat{C} \le C$ which proves the second point.

□

Theorem 5 *For a pseudo-ranked tree with $\hat{c}_i^+$ calculated by step 5 scheme $\widehat{S}$ based on a given B+ tree, the rejection rate and cost C incurred by algorithm 4 do not exceed those for algorithm 3 applied to the original B+ tree, if the same F is used in both algorithms.*

Proof: Let scheme S be the scheme defined in A/R simulation: $c_i^+ = F^{h(n_i)}$, $c_i^- = 0$. The rejection rate and cost C stay identical for algorithm 3 applied to the original tree and algorithm 4 applied to the S-defined tree. Let us use S as scheme S in step 5. Then $\hat{c}_i^+ = min(c_i^+, F^{h(n_i)}) \le F^{h(n_i)} = c_i^+$ because the same F is used in schemes $\widehat{S}$ and S. The theorem proof comes from applying lemma 5 to schemes $\widehat{S}$ and S.

□

6. Performance Evaluation

Further we will use the following notations:

RS - sampling from ranked trees,

NRS - A/R sampling from non-ranked trees,

PRS - sampling from pseudo-ranked trees.

For the PRS method, all bounds will be calculated based on steps 1-5.

We first notice a strict ordering of update overhead (u.o.) and rejection rate (r.r.) incurred by three sampling methods:

$1 >$ RS u.o. $\ge$ PRS u.o. $\ge$ NRS u.o. $= 0$,

$0 =$ RS r.r. $\le$ PRS r.r. $\le$ NRS r.r $\le (F/2)^{h(T)} - 1$,

assuming that $f(n) \ge 2$ for all non-leaf nodes n and that F is common in PRS and NRS methods. Here *update overhead* is measured as the average number of bound-maintenance I/Os per a single insert /delete-related I/O, the root node is assumed to be a memory resident.

In addition to superseding RS and NRS in terms of u.o. and r.r. respectively, the PRS method can gradually approach and fully simulate the lowest (zero) RS rejection rate and the lowest (zero) NRS update overhead. The complete tunability of "pseudo-ranked" sampling method is illustrated in the following table obtained from experiments run on a prototype software.

PRS	Q=1 u.o.	r.r.	0.3 u.o.	r.r.	0.1 u.o.	r.r.	0.03 u.o.	r.r.
A=10	0%	907.7	0.01%	35.56	0.18%	9.874	0.43%	6.182
3	0.06%	114.6	0.22%	4.208	1.12%	2.326	3.06%	1.900
1	0.60%	9.837	1.22%	0.962	4.14%	0.671	6.64%	0.579
0.3	4.18%	1.047	6.25%	0.267	15.2%	0.211	20.1%	0.188
0.1	12.1%	0.326	18.3%	0.104	42.1%	0.085	49.3%	0.078
0.03	22.9%	0.105	39.4%	0.040	75.4%	0.033	78.0%	0.031
0	78.0%	0	78.0%	0	78.0%	0	78.0%	0

Pseudo-ranked trees are built by 100,000 random inserts based on a B+ tree with h(T)=5, F=Fmax=34, Favg=10.72 .

Table 6.1: PRS Performance Tunability

In table 6.1, the extreme cases are reached with $A = 10, Q = 1$ (NRS simulation) and with $A = 0$ (RS simulation.) A desired balance between the two penalties, u.o. and r.r., is controlled by tuning parameter A. Finer tuning of balance and a simultaneous u.o./r.r. decrease can be achieved by selecting parameter Q. But the most striking result is that in $A = 1, Q = 0.3$ region both penalties are kept very low: about 1 reject with 1% update overhead. In the following table we compare performance of three sampling methods within this region for different tree heights and fanout distribution factors.

c(T)	h(T)	F/Favg	RS u.o.	RS r.r.	PRS u.o.	PRS r.r.	NRS u.o.	NRS r.r.
1,000,000	6	30/11	80.79%	0	0.90%	0.950	0%	1457.0
1,000,000	5	50/16	80.18%	0	0.56%	0.901	0%	936.5
1,000,000	4	80/23	77.78%	0	0.40%	0.877	0%	490.5
100,000	3	80/28	71.44%	0	0.33%	0.928	0%	65.6
10,000	2	80/43	64.90%	0	0.52%	0.863	0%	6.0

Pseudo-ranked trees with A=1, Q=0.3 are built by random inserts.

Table 6.2: Performance Comparison

Experimental results in table 6.2 demonstrate a clear superiority of the PRS method over each of the two other methods. In addition, sampling from pseudo-ranked trees exhibits a very high rejection rate stability over arbitrary tree sizes and heights. The update overhead also stays stable and low.

It is important to keep in mind that frequent updates of high-level tree nodes contribute to frequent locking of big portions of the tree, thus increasing dead lock rate in a multi-user environment. The RS method locks the tree root with every insert/delete operation, that alone can make it impractical. Our method, on the contrary, does fewer rank-related updates at higher tree levels and therefore curbs the contingency problem expansion.

If parameter Q is set below 1, it not only affects rejection rate reduction, but also forces the majority of rejects to occur at low tree levels. Because of this, a number of nodes processed during a single descent tends to stay at or close to a tree height, no matter whether the descent attempt is rejected or accepted. Therefore, assuming that the root is cached, the cost of obtaining a single sample can be approximated by $cost \approx (r.r.+1)(h(T)-1) = \frac{r_f^+(T)}{c(T)}(h(T)-1)$. This estimate is conservative, i.e. it is always higher or equal to the precise cost $C = \frac{\sum_{(N,n_i)} c_i^+(N)}{c(T)}$. The ingredients of the approximate cost formula are usually available at the tree root, including a tree cardinality estimate (which plays an important role in query optimization.) It gives us an opportunity to calculate the sampling cost before sampling and retrofit this data into the query optimizer, which in turn might pick a better index if several indexes are available.

In this paper we used the exact formula $r.r. = F^{h(T)} f(T)/c(T) - 1$ for calculating NRS rejection rate. The obtained precise rates differ from approximate rates $r.r. \approx (F/F_{avg})^{h(T)} - 1$ we would have obtained if we took the estimation approach presented in [OlRo89]. The F/F_{avg} ratio usage tends to deliver overoptimistic estimates, since between the two fanouts $f^- = F_{avg} - \delta$ and $f^+ = F_{avg} + \delta$ with the same deviation δ from the average, f^- contributes to an r.r. increase stronger than f^+ contributes to an r.r. reduction. This imprecision is very small for fan-balanced B+ trees and is about three times overoptimistic on real space-balanced 100K-1M trees according to our observations.

Apart from performance measuring, cardinality bounds contained in pseudo-ranked trees allow for efficient estimation of restriction selectivities, even without random sampling. Selectivity measurement is usually reduced to estimation of cardinalities of several node branches at the level where a given restriction range covers more than one tree node. The branch cardinality estimate of node n_i of parent N can simply be obtained as $c_i^o(N)$ stored in N node. We are also assured that the real branch cardinality $c(n_i)$ is always within the bounds $c_i^o(N)/\prod_{i=0}^{h(n_i)-1}(1+AQ^i)$, $c_i^o(N)\prod_{i=0}^{h(n_i)-1}(1+AQ^i)$

For $Q < 1$, these bounds in turn have some height-independent bounds due to the product $\prod$ convergence. In the $A = 1, Q = 0.3$ case, the bounds are:
$0.319c_i^o < c(n_i) < 3.13c_i^o$
regardless of the branch height. These bounds are very strong when applied to space-balanced trees, considering the fact that they are valid for any branch, including a whole tree, and keeping in mind that just a single node fanout is typically limited by similar bounds, see tables 3 and 6.2.

7. Conclusions

We described a new sampling method, the performance of which on industry-standard space-balanced B+ trees is close to the fastest "one descent" sampling. Both sampling cost and tree maintenance overhead are small, stable over a variety of the tree shapes and sizes, and tunable to arbitrary tree update and sampling speed demands. This new method unifies the existing "ranked" sampling and "A/R non-ranked" sampling approaches at the technological level, simulates each of them by simply setting A and Q parameters to boundary values, and supersedes each of them with A and Q set between the boundaries.

In the future we consider an investigation of how partial randomness of fanout distribution, present in all B+ tree maintenance procedures, can contribute to further improvement of sampling performance.

Acknowledgments

I would like to thank Christoph Freytag, Richard Helliwell, Zia Mohamed, and Rick Anderson for valuable comments on the content and practical applicability of this paper.

REFERENCES

[Ark84] H. Arkin *"Handbook of Sampling for Auditing and Accounting,"* McGraw-Hill, 1984.

[BeKr75] B. Bennet and V. Kruskal "Lru Stack processing," *IBM Journal of Research and Development*, 19(4):353-357 (July 1975).

[CDRS86] M. Carey, D. DeWitt, J. Richardson, E. Shekita "Object and File Management in the EXODUS Extensible Database System," *Proceedings of the twelfth VLDB conference*, pp. 81-100 (August 1986).

[HOT88] W. Hou, G. Ozsoyoglu, B. Taneja "Statistical Estimators for Relational Algebra Expressions," *Proceedings of the ACM PODS*, pp. 276-287 (March 1988).

[HOT89] W. Hou, G. Ozsoyoglu, B. Taneja "Processing Aggregate Relational Queries with Hard Time Constraints," *Proceedings of the ACM SIGMOD Conference*, pp. 68-77 (June 1989).

[Knu73] D. Knuth *"The Art of Computer Programming: Vol. 3, Sorting and Searching,"* Addison-Wesley (1973).

[LiNa89] R. Lipton, J. Noughton "Estimating the Size of Generalized Transitive Closures," *Proceedings of the fifteenth VLDB conference*, pp. 165-172 (1989).

[LiNa90] R. Lipton, J. Noughton "Query Size Estimation by Adaptive Sampling," *Proceedings of the ACM PODS*, (March 1990).

[LNS90] R. Lipton, J. Noughton, D. Schneider "Practical Selectivity Estimation through Adaptive Sampling," *Proceedings of the ACM SIGMOD Conference*, pp. 1-11 (May 1990).

[LWW84] H. Lenz, G. Wetherill, P. Wilrich, editors *"Frontiers in Statistical Quality Control 2,"* Physica-Verlag, Wurzburg, Germany, 1984.

[MuDe88] M. Muralikrishna and D. DeWitt "Equi-depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries," *Proceedings of the ACM SIGMOD Conference*, pp. 28-36 (June 1988)

[OlRo86] F. Olken and D. Rotem. "Simple Random Sampling for Relational Databases," *Proceedings of the twelfth VLDB conference*, pp. 160-169 (1986).

[OlRo89] F. Olken and D. Rotem. "Random Sampling from B+ Trees," *Proceedings of the fifteenth VLDB conference*, pp. 269-277 (1989).

[PiCo84] G. Piatetsky-Shapiro and C. Connell "Accurate Estimation of the Number of Tuples Satisfying a Condition," *Proceedings of the ACM SIGMOD Conference*, pp. 256-276 (June 1984).

[SrLu88] J. Srivastava and V. Lum "A Tree Based Access Method (tbsam) for Fast Processing of Aggregate Queries," *Proceedings of the 4th International Conference on Data Engineering*, pp. 504-510, IEEE Computer Society (1988).