Dynamic Query Optimization in Rdb/VMS

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

Abstract

This paper addresses the key theoretical and practical issues of dynamic query optimization - a strategic direction in achieving high and reliable Rdb/VMS* performance. The basic concepts, operational structures, and dynamic execution metrics have been available to the user community since version 4.0 (1990). This article describes for the first time the underlying reasoning which cements these concepts and which is based on extensive research and prototyping.
The dynamic query optimization mechanics are described in concert with explanations of why and how certain arrangements contribute to a given optimization goal. Compared to the traditional approaches, dynamic query optimization offers a much more realistic view of cost distribution modeling and present a new competition-based architecture capable of resolving the major limitations of static optimization. In particular it addresses the problems of data skew, cost function instability, and host-variable sensitivity.

1. Introduction

The overwhelming success of the relational model over the CODASYL and structured approaches in the 1980s had a single major reason behind it. Relational database engines took a heavy burden of manual design and coding of query execution plans away from the user. In a majority of relational run times today, creation of a good execution plan is done by generating a number of possible execution plans, estimating their costs, and selecting the lowest cost plan.

With growing database sizes and increased query complexity in the 1990s, the compile-time optimization approach exposed suboptimality and instability to such a degree that many vendors provided the users with means of manual plan specification and plan "freeze" capabilities, i.e. partially returned the optimization burden back to the users. Causes for suboptimality and instability can be traced to two major limitations of the traditional optimizers.

Cost function instability: The cost estimation error grows rapidly with the size of relational expressions. Specifically, the cardinality and cost estimation error grows exponentially as the number of joins increases [IoCh91]. This causes suboptimal plan selections once queries become complex and deviate from the straightforward usage of chains of "unique index" joins. No systematic approach is known today for solving this problem.

Sensitivity to variables: With multiple runs of an execution plan or with iterative execution of query subplans, a number of variables can change their values between different runs and iterations: host-language variables, iteration context, system parameters like amount of available space, concurrency rate, etc. With a skewed data distribution, sensitivity to variables amplifies. Such a "parametrical" nature of query execution limits the performance of a single plan and calls for dynamic changes of execution strategy based on changing execution parameters [GrWa89],[INSS92].

Cost function instability is a severe problem. Ioannidis and Christodoulakis [IoCh91] demonstrated that the cardinality error of n-way join grows exponentially with n even if we have good estimates of the number of records delivered by the table scans. It implies that the cost of different execution plans calculated with the industry-standard cost model [SACL79] often has no validity, and thus, the "best" query execution plan is picked quite at random on the scale of real execution costs.

One way of improving execution plan selection in the big estimate error cases is to generate several plans with relatively low costs at random, then actually execute these plans and pick the best one. This approach, however, is not applicable to ad hoc queries (since these are executed only once) and, more importantly, does not address sensitivity to variables.

To find a better approach, we extended a study of error propagation to cover: (a) four major operators AND, OR, NOT, and JOIN, (b) real shapes of probability distribution, not just their variances, (c)

* Rdb/VMS is a trademark of Digital Equipment Corporation.

1063-6382/93 $03.00 © 1993 IEEE

arbitrary correlations between the operand distributions, (d) a mixture of different correlations, aiming to explore a typical case of unknown correlations.

The results of this study reveal that the expressions based on the four operators above, applied to data with unknown correlations between columns, tend to produce the following effect on cost estimation:

(A) They increase the variance, and so flatten the distribution concentrations around the means discovered by estimation procedures, i.e. they negate estimation precision.

(B) They concentrate 50% of the distribution in a small area around zero and spread the other 50% into a broad adjacent area when the proportion of ANDs and JOINs dominates ORs as described in Section 2.

(C) If OR operators dominate, a mirror-symmetry of (B) results: the 50% of the distribution is concentrated at the highest cost point, and the remainder is spread in a broad range near it.

(D) The result distributions are well approximated by truncated hyperbolas in disbalanced AND/OR cases, with hyperbola skewness quickly increasing when disbalance increases.

These results match our production experience, although we observed that concentration of a high probability around zero occurs much more often than a concentration around the highest cost: reflecting the fact that the real queries rarely return the whole database. Our modeling of probability distributions proved that estimation precision of JOIN and logical operators doesn't just deteriorate, it degenerates to the most uncertain, uniform and hyperbolic distributions.

We found that because of the quick degradation of estimation precision, it is better to abandon the idea of following a single execution plan, and instead, to use a set of local plans for the reliably-estimated (or atomic) query subexpressions. Execution of the local plans increases estimation precision of bigger query subexpressions and thus facilitates merges of local plans into bigger plans until the entire query is resolved.

Based on the hyperbola model, we found that it is cost-effective to run several local plans simultaneously with the proportional speed for a short time, and then select one "best" plan and run it for a long time. During the local plan run, it may become advantageous to abandon the local plan execution in the middle and switch to a better alternative.

These findings give hope that a new query processing architecture can solve the problems of the cost function instability and sensitivity to variables inherent in static plan selection. Such new architecture should provide for dynamic strategy switching, should intensively use dynamic estimation, and should employ a dynamic cost model to control execution direction.

In this paper we describe an architecture and operational principles of the **dynamic optimizer** of a single table access component based on the ideas discussed above, and implemented in Rdb/VMS relational database management system in 1990.

After a few years of production experience and improvement, we observed a drastic relaxation of the traditional optimizer limitations for single table access. The problem of incorrect strategy selection is largely gone, and part of it is transformed into a smaller problem of reducing the overhead of parallel strategy runs and of unsuccessful (abandoned) runs. The problem related to a parametric query nature, is fully solved for user variables and iteration context. It is only partially solved for caching disk pages since the prediction is obstructed by interference of other queries or different branches of the same query.

We will present here the results of the distribution study and a competition model of controlling the local plan execution arrangements. These arrangements are grouped into four major competition tactics. Additional information on these tactics along with examples of strategy switching can be found in [Ant91A]. More tactics and a complete theoretical account of competition within the dynamic optimization framework is given in [Ant91B] research report. An overview of dynamic optimization as part of Rdb/VMS performance features is presented in [HoEn91].

The rest of this paper is organized as follows. Sections 2 and 3 cover the probability distribution study and a competition model. Sections 4-7 concentrate on a single table retrieval optimization with basic arrangements presented in Section 4, initial estimation described in Section 5, joint scan strategy explicated in Section 6, and major competition tactics highlighted in Section 7. Section 8 concludes the paper.

2. Transformation of Data Distribution

Within the scope of a single table access, knowledge of Boolean restriction selectivity contributes the most to optimization of the retrieval process. Given a table cardinality c and amount r of records for which Boolean evaluates to TRUE, we call **selectivity** a proportion $s = \frac{r}{c}$ of restricted records, $0 \leq s \leq 1$. At retrieval time, selectivities of a Boolean

or some of its subexpressions might be known, estimated with some precision, or totally unknown. Any of this knowledge can be represented as a selectivity probability density function (or simply a selectivity distribution).

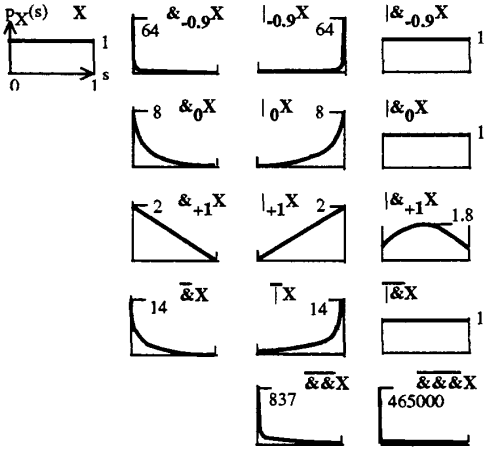
If X is a predicate on attributes of table T, e.g. $X \equiv (T.x < 10)$, let $p_X(s)$, $s \in [0,1]$ be a probability density function of selectivity s, $\forall s\ p_X(s) \geq 0$, $\int_0^1 p_X(s)ds = 1$. A negation $\sim X$ of X evaluates to TRUE on all T records where X does not evaluate to TRUE, hence $\forall s\ p_{\sim X}(s) = p_X(1-s)$, that is to say that $p_{\sim X}$ is a mirror symmetry of p_X.

ANDing $X\&Y$ of predicates X, Y evaluates to TRUE on intersection of subsets of T for which X and Y evaluate to TRUE. If we split $[0,1]$ into small intervals I_i, surrounding points $i \in [0,1]$, with small integrals $W_i = \int_{I_i} p_X(s)ds$ of p_X on I_i, and also do a split $\{I_j\}$, with integrals W_j for p_Y, then in the area surrounding $i * j$, a probability weight $W_{i,j}$ of $p_{X\&Y}$ is going to be $W_i * W_j$, assuming independence of X and Y. In other words, we first transform p_X, p_Y into two groups of single weighted point estimates, then calculate $\dot{p}_{X\&Y}$ points and weights for all combinations of i and j points, and then convert a "point/weight" $\dot{p}_{X\&Y}$ version into an approximate probability density function $\tilde{p}_{X\&Y}$. Further experiments are all based on numeric computations, like above, so we will loosely use $p_{X\&Y}$ notation instead of $\tilde{p}_{X\&Y}$.

ORing $X|Y$ can be expressed as $\sim(\sim X \& \sim Y)$ and thus is reduced to AND case: $p_{X|Y}$ is a transformation of p_X, p_Y mirror-symmetrical to $p_{X\&Y}$.

If for a pair of subsets of T with selectivities s_X, s_Y we pick only the subset pairs with the biggest intersection, then on such subset pairs $s_{X\&Y} = min(s_X, s_Y)$. Selectivity distribution $p_{X\&Y}$ calculated under this assumption is said to be calculated under assumption of +1 correlation between predicates X, Y. Similarly, $p_{X\&Y}$ under -1 correlation assumption is based on the smallest subset pair intersections with $s_{X\&Y} = max(0, s_X + s_Y - 1)$. For any assumed correlation $c \in [-1,+1]$, $s_{X\&Y}$ is linearly interpolated between the points -1, 0, +1 with values $max(0, s_X + s_Y - 1)$, $s_X * s_Y$, $min(s_X, s_Y)$.

We will use $X\&_c Y$, $X|_c Y$ notations, $c \in [-1,+1]$, to specify the assumed correlation. If we assume a mixture of assumed correlations c with a uniform probability of c on $[-1,+1]$, we will talk about an unknown correlation assumption, and will use the notations $X\overline{\&}Y$, $X\overline{|}Y$. Finally, unary $\&X$, $|X$ operations will be a shorthand for $X\&Y$, $X|Y$ in cases when $p_X \equiv p_Y$. Figure 2.1 below exposes the shapes of transformations of ANDs, ORs, and some AND /OR combinations applied to uniform selectivity distributions under different correlation assumptions.

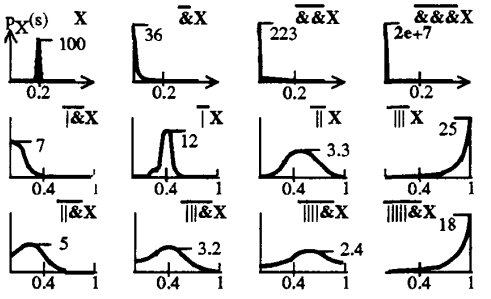


Here selectivity distributions are shown
for different Booleans applied to predicate X
with correlation assumptions:
+1 (e.g. in $\&_{+1}X$), 0, -0.9, and "unknown" ($\overline{\&}X$)
with notations $\&X$, $|X$ used for $X\&Y$, $X|Y$
where X and Y have the same distribution.

Figure 2.1: Transformation of Uniform Distributions

Distributions in Figure 2.1 illustrate several important points. Application of several ANDs or several ORs to uniform distributions produces "crescent", "triangle", and L-shaped distributions, with their skewness quickly increasing upon correlation decrease, and upon adding more operators of the same kind. A mixture of equal numbers of ANDs /ORs restores the original symmetry to the resulting symmetry of uniform or near uniform distribution. All asymmetrical transformations of uniform distribution are well approximated (but not fully matched) by truncated hyperbolas. For instance, truncated hyperbolas fit $\overline{\&}X$ with relative error 1/4, $\overline{\&\&}X$ with error 1/7, $\overline{\&\&\&}X$ with error 1/23. Here relative error of hyperbola $h_X(s)$ fitted to $p_X(s)$ is $max_s|p_X(s) - h_X(s)|/(max_s p_X(s) - min_s p_X(s))$.

Uniform selectivity and its AND/OR transformations reflect the cases of very high uncertainty: note that half the probability is concentrated along each of horizontal/vertical hyperbola legs (when abscissa and ordinate are depicted symmetrically). The cases of various degrees of uncertainty, typically represented as a bell-shape around the estimation point, and their AND/OR transformations are illustrated below in Figure 2.2.



Here selectivity distributions are shown
for AND/OR chains, assuming unknown correlation,
applied to estimation p_X of predicate X
with mean m=0.2 and error e=0.005.
Same notations are used as in Figure 2.1.

Figure 2.2: Degradation of Certainty

Under the unknown correlation assumption, application of ANDs/ORs to bell-shaped or single point distributions trigger several processes whose effect can be observed in Figure 2.2. If the distance of "bell" from any end of $[0,1]$ interval is much larger than the bell spread (standard deviation), a single AND or OR instantly increases a distribution spread, making it of the same order as the distance (see $\overline{\&}X$, $|X$ cases). Once distribution becomes a bell located nearby or touching one interval end, say the zero end, then ANDing converts it to an L-shape and ORing flattens it and spreads away from zero end, about doubling the original spread. When bell spreading toward the interval center repeats and reaches the center, the next application of the same operator produces an L-shape (see $\overline{|||}X$, $\overline{|||||}\&X$ cases). When the bell is near the left end (or right end, or center), repetitive ANDing (or ORing, or either) produces L-shapes with quickly increasing skewness.

Summarizing the processes above, we can state:

(1) An estimation precision relative to the closest distance from the interval end is instantly nullified by a single ANDing or ORing.

(2) An absolute precision is reduced and converted to an end-to-end-bell uncertainty by applying enough ORs and ANDs, whichever spreads the distribution toward the center.

(3) Except for some specific AND/OR combinations producing bell shapes as in (1),(2), other AND/OR combinations produce L-shapes with their skewness increasing with AND/OR disbalance.

The effect of ANDing/ORing of predicates with different distributions is largely the same as in the cases above. The JOIN operator behaves almost identically to the AND operator when multiple joins use the same key which is unique for all underlying tables - the key domain cardinality should be used in the selectivity definition. A general JOIN case produces uncertainty even faster, but still requires a more detailed investigation.

We can state now that regardless of the presence or precision of estimation on underlying predicates, intermediate selectivity distributions of join /select queries are predominantly Zipf-like [Zipf49]. Therefore, the traditional compile-time optimizers are largely indiscriminating in choosing an execution plan, and there is no hope to approach optimal query performance unless by arranging and running partial plans, improving estimates dynamically, and using fresh estimates to extend the plans until the entire query is resolved.

3. Competition Model

Suppose, there are two alternative plans A_1 and A_2 aiming to achieve the same goal, and we know that the cost of both plans have L-shaped distributions with 50% probability concentrated in small cost regions $[0, c_1]$ and $[0, c_2]$ and 50% probability widely spread to the right of them, with mean costs M_1, M_2 ($c_1 \ll M_1 \leq M_2$, $c_2 \ll M_1$), and with mean cost m_2 of A_2 on $[0, c_2]$. The traditional optimizer would choose plan A_1 (since $M_1 \leq M_2$) and run it to the end incurring average cost M_1.

A better arrangement is to run A_2 till the cost reaches c_2 and then switch to A_1. With 50% chances, A_2 completes first, incurring an average cost m_2. Otherwise, the combined cost of both plan runs has an average cost $c_2 + M_1$. Putting together the weighted costs of the two scenarios, we come up with an average cost $(m_2 + c_2 + M_1)/2$, about twice smaller than the traditional M_1 because $m_2 \leq c_2 \ll M_1$. If both L-shapes are truncated hyperbolas, a still better approach is to run both plans simultaneously with some proportional speeds, and switch to plan A_1 at some optimal point. We call the arrangements above a **direct competition** between the alternative plans.

Suppose that plan A_2 breaks down into two consecutive stages A' and A'' with the A' cost much lower than the A'' cost, and with a reliable cost estimator of A'' available at stage A'. We can run A' first and do a continuous recalculation of the A'' cost probability function based on the A'' cost estimator, so that the A'' cost uncertainty decreases. At each point of A' we compare A_1 and fresh A'' cost distributions, and switch to A_1 or continue based on some probabilistic cost model. We call this arrangement a

two-stage competition. Note that for this competition to be effective, an L-shape assumption of A_1, A_2 cost distributions is no longer necessary.

Precise definitions of both types of competition, their cost model, and examples of numeric cost evaluation are given in [Ant91B]. In the following sections we describe a new architecture of the Rdb/VMS single table retrieval component which deals with L-shaped distribution dominance directly by using several competition-based tactics.

These tactics deal not only with uncertainties of intermediate results, but also with several other important uncertainty cases.

(a) A retrieval process can be terminated before completion (see fast-first optimization goal in Section 4). This imposes an "external" selectivity which typically has an L-shaped distribution.

(b) Some indexes or index portions can have their sequence coincided to a various degree with physical record locations. This clustering effect may not be known or may be hard to detect, so it adds a significant uncertainty to the cost estimation.

(c) Even if a single column selectivity is estimated with good precision and inexpensively, the actual cost of index scan and data record fetches measured in physical I/Os is often unpredictable because the pattern of caching the disk pages is influenced by many asynchronous processes totally unrelated to a given retrieval.

4. Single Table Access Optimization

When for a given table one or more indexes are available with some restrictions specified on them, optimization of this table access can involve various sophisticated techniques, having a tremendous impact on performance. However, the task of choosing a correct retrieval strategy can be difficult even in simple cases. Consider the next to simplest query

*select * from FAMILIES where AGE >= :A1;*

with parameter :A1 taking values 0 and 200, delivering all or no records in two different runs. In this case, a correct choice between the sequential ($\geq$0) and index ($\geq$200) retrieval strategies can only be done dynamically on a per-run basis, using either the dynamic estimation or competition method.

The available indexes are used in several different ways during retrieval. If a certain retrieval order is requested, some indexes can provide this order. We will call them **order-needed** indexes. If an index contains all attributes needed for table restriction evaluation and for retrieval result delivery, the index scan alone can select and deliver all result records. We will call such indexes **self-sufficient** indexes, as opposed to **fetch-needed** indexes which would require data record fetches.

With several self-sufficient indexes present, the only optimization task to be resolved is to pick the one whose scan is the cheapest. Quite to the contrary, several fetch-needed indexes are used most optimally by their collective (joint) scan aiming at delivery of the shortest list of record IDs (RIDs) satisfying a cumulative fetch-needed index restriction. The RID list is built by intersecting /unionizing individual index RID lists according to the restriction AND/OR operations, and then is used for final fetches of data records. Alternatively [Babb79],[MoHa90] or complementarily [Ant91B] to RID lists, the bitmaps can be used for AND/ORing during joint scan. (See the detail joint scan discussion in Section 6.)

All together four basic retrieval strategies are widely used today:

Tscan: Full table scan (no indexes involved) - a classical sequential retrieval.

Sscan: Self-sufficient index scan.

Fscan: Fetch-needed index scan with immediate data record fetches - a classical indexed retrieval.

Jscan: Joint scan of fetch-needed indexes.

It is gradually becoming common knowledge in the industry that the static optimizer is helpless to reliably choose a correct scan strategy if host-language variables are present or when a good data distribution/interaction estimation is not possible or too costly. Dynamic reevaluation of execution plan [GrWa89] helps only partially since some estimations are impossible, or imprecise, or too costly when done at the start retrieval time, and since data interaction uncertainty can often be unresolvable unless by the actual retrieval run.

In Rdb/VMS we combine dynamic estimation and plan reevaluation with a competition of different scans in highly uncertain areas. With this new approach we are able to optimize retrieval for two distinct performance goals: a reduction of total retrieval time and speeding retrieval of the first few result records. The correct setting of these **total-time** and **fast-first** optimization goals by the optimizer or via user request improves query performance up to a few decimal orders.

Via extended SQL syntax, a user can explicitly state his optimization request as OPTIMIZE FOR FAST FIRST (or TOTAL TIME). Suppose that a query execution plan contains any of EXISTS, LIMIT TO n ROWS, SORT, COUNT or other aggregate nodes. For a given retrieval node, the static optimizer searches the plan to see what node from the above list immediately controls the retrieval node. If EXISTS or LIMIT TO node controls the

retrieval node, the fast-first retrieval optimization is requested. A detection of the SORT or aggregate control sets the total-time optimization request. Otherwise, the user-defined or default optimization goal is used.

For example, the optimization goal in
*select * from A where A.X in (*
select distinct Y from B where B.Y in (
select Z from C limit to 2 rows))
optimize for total time;
will be set to
fast-first for table C because of "limit to",
total-time for B because of SORT needed for "distinct",
total-time for A because of explicit cursor request.

Among the retrieval strategies, T-, S-, and Fscans are naturally suitable for fast-first optimization because they can deliver records immediately while scanning. Jscan, on the other hand, is good for total-time retrieval because it builds an RID list off line. This way the Jscan optimization mechanisms can greatly benefit from using all available memory and yet the same mechanisms release the memory for other uses before any records are delivered.

For a given optimization goal, a single scan strategy or a combination of strategies is determined either statically or dynamically at start retrieval time. Static optimization covers such clear cases as selection of Tscan with absence of indexes or selection of Sscan if only one useful index is available and this index is self-sufficient. When the choice of scan is not clear, the dynamic optimizer tries to resolve it by doing inexpensive estimates of scan costs based on parameter values and the current state of data distribution. The strategy choice may be resolved unambiguously, like in choosing the lowest cost Sscan among available self-sufficient indexes. But often, a substantial uncertainty or impossibility of estimation opens the way for better tactics of competition and cooperation of more than one strategy.

We can now draw a flowchart of a single table retrieval subsystem capable of exercising different competition tactics.

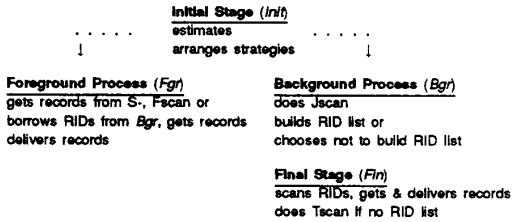


Figure 4: Flowchart of Single Table Retrieval

The foreground (*Fgr*) and background (*Bgr*) processes can run individually or simultaneously, working for a common goal. *Fgr* delivers data immediately (satisfying the fast-first goal if stated) and is capable of completing the entire retrieval by itself. *Bgr* does its best to come up with the shortest possible RID list or to recommend Tscan usage at the final stage (*Fin*). *Fin* is executed only upon *Bgr* completion as an alternative to the *Fgr* records delivery. *Fgr* and *Bgr* cooperate by exchanging data, for instance, *Fgr* may borrow RIDs from *Bgr* in order to satisfy a fast-first request.

5. Initial Stage

The initial retrieval stage arranges the available useful indexes into single or combined scan strategies. This includes picking the best self-sufficient index for Sscan or finding a good arrangement of indexes to be involved in Jscan. All initial stage decisions are based on estimates made with current parameters, data distribution, and optimization goals in mind. In addition, the estimation phase should be significantly shorter than the productive retrieval phases.

Estimation of the number of RIDs satisfying a given index restriction contributes the most to the correct initial arrangement. A widely known estimation method based on storing the column distribution histograms unfortunately has several major drawbacks. It fully depends on costly data rescans for histogram maintenance, and it can only be used for range-producing restrictions. But even for range estimates, histograms fail to detect small ranges falling below granularity, though the smallest ranges must be detected and scanned first, often without looking at bigger ranges.

In Rdb/VMS we employ an index B-tree as a hierarchical histogram for range estimation. Suppose we want to estimate RIDs in the AGE range [30:32] using the index B-tree below.

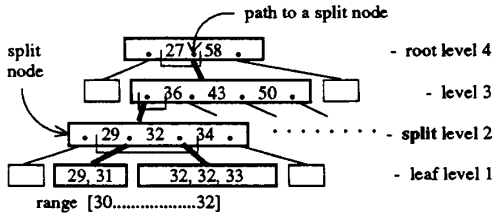


Figure 5: Estimation by Descent to a Split Node

We first descend the tree from the root along the path containing only those nodes which branches include all range keys. The lowest node of the path is a "split" node. Its level is a "split" level l. The number of its neighboring children containing the range is $k+1$ if $l > 1$, and the number of range-satisfying RIDs is k if $l = 1$. Assuming that the left- and rightmost children of the split node range contain 50% of range-satisfying keys (and thus counting those two nodes as one) and assuming the average tree fanout be f, we can now estimate the number of range RIDs as

$$RangeRIDs \approx k * f^{l-1} .$$

(In the above example $l = 2$, $k = 1$, and, assuming $f = 3$, $RangeRIDs \approx 1 * 3^{2-1} = 3$.)

This "descent to split node" method is fast, well suited for small ranges, and its estimate is always up-to-date. More precise estimation would require a good inexpensive random sampling on range children of a split node. Random sampling can estimate RIDs with any restrictions, including pattern matching, complex arithmetic, comparing attributes of the same index. We have recently developed a new inexpensive sampling method [Ant92] which significantly supersedes the known acceptance/rejection method [OlRo89] and is tuned for heavy usage within the dynamic optimization framework.

When we use the range estimates for arranging the best order of fetch-needed indexes involved in Jscan, we apply several techniques to reduce the estimation cost when very inexpensive strategies are available. The indexes to estimate are prearranged in the most probable ascending RID quantity order. The freshly (and optimally) reordered indexes are used for the next retrieval estimates as a starting point. If a very short range is discovered (which typically happens right away because of preordering), the initial stage estimation terminates immediately to save on estimation cost. In addition, an empty range detection cancels all retrieval stages and delivers the "end of data" condition at once. These techniques are instrumental in achieving high performance of short OLTP transactions.

6. Jscan - Joint Scan Strategy

Suppose that we have one or more fetch-needed indexes available for no-order-needed retrieval with total time to be optimized and with all index-bound restriction portions connected by ANDs. We can then scan these indexes collectively in order to produce a reasonably short intersected RID list or to reliably determine that a sequential table scan (Tscan) is the best retrieval option. Thus defined, Jscan performs optimally when (1) a correct subset of indexes is selected (empty subset yielding Tscan) and (2) the right sequence of scanning the selected indexes is determined.

As described in Section 5, the initial stage does its best in arranging a good ordered index subset. Sometimes, for instance in shortcut cases, the initial stage resolves the ordered index subset unambiguously. But often the initial arrangement is optimal only with some probability and leaves room for competition to skip the truly unproductive indexes and to partially reorder the sequence of remaining indexes.

Method. Jscan scans preselected indexes in the best prearranged order, i.e. roughly in the ascending selectivity direction. Each index scan produces a RID list, stores it into a main memory buffer, and writes it into a temporary table upon buffer overflow. Each non-last index scan also produces a filter to assist a RID list intersection: an in-buffer sorted RID list or a hashed in-memory bitmap [Babb79] for temporary tables. While building a non-first RID list and filter, all candidate RIDs falling outside of the previously completed filter are eliminated, so that each new RID list becomes an intersection of the current and all previous lists. The non-competitive indexes are detected during scans and their incomplete RID lists and filters are discarded. As a result, either the last complete RID list is delivered or absence of such indicates Tscan optimality.

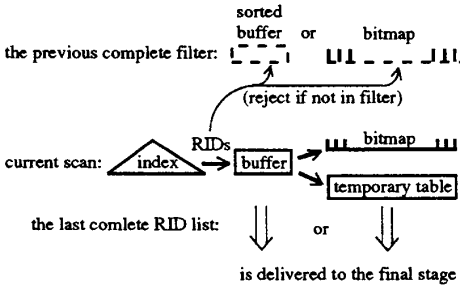


Figure 6: Jscan Flowchart

In this method, the unproductive scans are eliminated by a two-stage competition. At any given point of Jscan, we define the guaranteed best retrieval process as either Tscan or retrieval by a complete RID list when such a list becomes available: this process is the second (final) retrieval stage performed after Jscan. During each index scan (the first stage, typically 10-100 times cheaper than the second stage), the cost of the final RID list retrieval can be reliably estimated from the current RID list.

The scan is terminated and discarded when the projected retrieval cost approaches (e.g. becomes 95% of) the guaranteed best retrieval cost.

An index scan, as the first stage of RID list retrieval, competes with the guaranteed best retrieval by means of actual scan run. Since the scan cost is typically only a fraction of the entire retrieval cost, the "first stage" investment in uncertainty removal stays low and becomes part of the current RID list retrieval if the index is not discarded. When the projected second stage cost approaches the guaranteed best cost, this indicates a failure of the current competition attempt, and we terminate the scan a bit before the costs are equalized.

The described criterion of strategy switching includes only a projected second stage cost. But occasionally, when a large portion of RIDs gets rejected by a filter or a local restriction, the index scan cost may dominate the already small guaranteed best cost. We handle this case by extending the strategy switch criterion with an index scan cost limit set to some proportion of the guaranteed best cost. This criterion portion controls the direct competition of the first (index scan) stage against the second (final retrieval) stage.

In addition to eliminating index scans, Rdb/VMS can partially change the order of index scans by limited simultaneous scanning of two adjacent indexes. The first scan to complete is the one which delivers a new RID list and filter. There is almost no overhead involved in simultaneous scanning because both indexes are to be scanned anyway. But with detection of a smaller RID list first, the subsequent filtering becomes more efficient. The simultaneous scan of this complementary and direct competition does not continue beyond the memory buffer since the cost of refiltering the partial RID list against the winning scan filter is low only within main memory.

A similar Jscan strategy with statically set thresholds controlling unproductive scan elimination was independently discovered and described in [MoHa90]. The statically-controlled Jscan, however, misses an opportunity to readjust to new, reliably determined, guaranteed best retrieval cost, nor can it reorder the scan sequence dynamically. But one ill-predicted alternative execution cost, when not corrected dynamically, can put further execution off-balance and make it suboptimal.

Yet another application of "engineering around the L-shape distribution" contributes to scan speed-up in Rdb/VMS. The RID list size quantity is split into several monotonically increasing regions. A zero-long RID list causes an immediate shortcut action. Lists up to 20 RIDs are stored in a small statically-allocated buffer, avoiding any run-time allocation and memory usage overhead. Bigger lists are stored in the allocated buffer. Even bigger lists flow into a temporary table and set the bits in a bitmap in which the size is as small as necessary for two-stage competition. Despite its simplicity, this "hybrid" scan arrangement is quite advantageous due to the underlying L-shaped distribution.

7. Retrieval Tactics

Background-Only Tactic

When total-time optimization is requested with no self-sufficient or order-needed indexes available, Jscan is almost always a strategy of choice over Fscan. Even if only one fetch-needed index is available, Jscan, unlike Fscan, runs a two-stage competition and might switch to Tscan whenever profitable. RID list accumulation in Jscan also opens the possibility for sorting the final RID list and thus accessing several records on a single page only once, not multiple times as in the case of random fetches. Running Jscan followed by the final retrieval stage involves only the background process, hence we call it background-only tactic.

Fast-First Tactic

Given the same scenario with the fast-first optimization goal, we will need a more sophisticated tactic for its resolution called here fast-first retrieval tactic. The Fscan's immediate record delivery is good in early termination cases but is inferior to Jscan if a forceful "close retrieval" comes late or does not come at all. Late termination happens for long scans with small-to-zero delivery or when after few first record acquisitions the user still wants to get the rest of the records. Therefore, it is much better to (1) run Jscan, (2) simultaneously run the foreground process which "borrows" RIDs from Jscan, fetches and delivers records, (3) optimize this parallel operation as a direct background/foreground competition.

Thus defined, the fast-first tactic combines the best of both worlds. If the records to be delivered are discovered quickly and the user is satisfied and stops retrieval without a big delay, the foreground process succeeds with no less speed than Fscan would. If, according to a competition criterion, the "fast-first" satisfaction becomes less realistic or less possible, the foreground process terminates and retrieval continues as in the background-only tactic with all the benefits of Jscan.

The foreground process of fast-first tactic not only provides for the record delivery, but also stores RIDs of all delivered records in its own buffer. Upon buffer overflow or upon a competition-controlled switch to the background tactic, the foreground run is terminated and the buffer is passed to the final

stage where it helps to filter out the already delivered records. The cost of the foreground buffer activities is marginal, and the cost of storing a RID list during Jscan is also marginal. In fact, the fast-first tactic has only one substantial overhead of fetching the records which are rejected upon total restriction evaluation. This overhead, together with a typical total uncertainty of the forceful termination point, leads to an L-shaped distribution of the foreground cost and finds its best resolution in a direct foreground/background competition.

Sorted Tactic

Suppose that for fast-first optimization only fetch-needed indexes are available and at least one of them delivers the requested order. We can run Fscan for the best order-needed index in a foreground and simultaneously run Jscan for the rest of the indexes in a background. Once Jscan produces a complete filter, this filter can be used to reject the Fscan RIDs before fetching records, evaluating a total restriction, and rejecting the same RIDs afterwards. This extra Jscan-supported filtering may eliminate a large number of record fetches that usually comprise the biggest cost portion of retrieval.

In this strategy arrangement, called Sorted tactic, a traditional order-delivering Fscan is supplemented with filter-delivering Jscan, making them cooperative parallel scans. When there is enough uncertainty in Fscan/Jscan cost distribution, a competitive parallel run of both strategies becomes a better arrangement compared to building Jscan filter first and running Fscan later. With this arrangement, a quick Fscan completion eliminates a potentially big Jscan overhead and vice versa, a quick Jscan filter delivery saves a potentially huge cost of unneeded Fscan fetches. We showed in [Ant91B] that under the assumption of hyperbolic distribution, the speed of Fscan/Jscan advancement should be proportional or equal for optimal competition performance.

With the sorted tactic, Jscan does not need to store RIDs into a temporary table since a bitmap (or a sorted RID buffer) filter is the only useful outcome. Nor does Fscan need to store RIDs in a foreground buffer since the foreground process delivers all the records to the end and the final stage (i.e. a potential foreground buffer user) is never executed.

Index-Only Tactic

If, in addition to fetch-needed indexes, some self-sufficient indexes are also available, each of them can directly compete with Jscan and between each other. Sscans are much "safer" retrieval strategies than Jscan because the worst Sscan scans one entire index whereas the worst Jscan scans all the available indexes plus may read all data pages (i.e. perform the most expensive retrieval operation.) Sscans are also much simpler than Jscan and thus allow for better estimation. In other words, comparing the cost uncertainties, Jscan has a much more skewed L-shape distribution making Jscan /Sscan competition more profitable than between-Sscan competition. This observation gives rise to another retrieval tactic called the index-only tactic (called this way because the results can be obtained by reading only indexes without touching actual records.)

The index-only tactic performs Sscan of the best self-sufficient index in parallel with Jscan. Sscan runs in the foreground delivering the results immediately and collecting RIDs in the foreground buffer to avoid duplicate delivery at the final stage if Jscan wins the competition. Upon foreground buffer overflow, Jscan terminates and Sscan continues because it is a safer strategy. If Jscan completes before competition is over with a small enough RID list, Sscan is abandoned in favor of a "sure" final stage RID list retrieval.

Other Tactics

All four tactics described above are implemented in Rdb/VMS V4.0. These tactics together with six more competition tactics are presented in [Ant91B] systematically covering the essential combinations of (1) fast-first/total-time optimization requests, (2) requests for specific order or absence of such, and (3) self-sufficient and fetch-needed index availability to cover requests 1 and 2.

Based on these practical and theoretical results, we view further development of a single table retrieval optimization as taking two different directions. First, we see a continuation of the refinement of the architecture along with the growing quantity and quality of tactics. Covering ORs and between-index subexpressions of table-wide Boolean expressions is a rich source for extending the tactics and the architecture. Second, we hope to achieve some drastic improvement in the way we conduct estimations. The advancement in the estimation area may resolve a significant portion of the uncertainties more efficiently than the actual competition run and thus contribute to simplification of the tactics and the architecture. Between the two tendencies toward sophistication and simplification, a new, better retrieval engine may emerge.

8. Conclusion

Since the conception of today's static optimizers with their cost minimization model based on mean-point estimation [SACL79], the precision and adequacy of this method has rarely been questioned. The most serious statement about this single point estimation imprecision was made recently in [IoCh91] by Y. Ioannidis and S. Christodoulakis who proved the exponential growth of the estimation error for increasing join size. When Rdb/VMS became one of the strategic company products and started to handle very large volumes of data, the severe limitations of a single point estimation approach became obvious to us and forced us to look for more adequate solution.

By means of statistical modeling, we discovered the dominance of the L-shaped distribution across all data flows circulating through the query execution graph. This not only explained the mean-point estimation imprecision, but also revealed the inadequacy of the whole static optimization approach. More importantly, by solving a cost minimization problem of dynamic strategy switches, we found an error-proof method of dynamic optimization that is based on and exploits the L-shaped distribution.

This new method, called competition, exhausts high-probability low-cost regions first - this often is done by simultaneous strategy runs, and it always switches to a better alternative - even in the middle of strategy execution. The competition method turned out to be so different that a complete rearchitecture of the query executor became necessary and was implemented in version 4.0 (1990) at the table retrieval level. As a result, the query executor became a highly sophisticated run-time component instead of just being an automata which passes records up the execution tree.

The production experience with two Rdb/VMS releases offered us an opportunity to observe a full range of strategy switches and combinations. This can be considered as (1) an empirical proof of correctness of the L-shaped distribution assumption and (2) evidence of competition model practicality. Through further extension and refinement of the dynamic optimization technology, we hope to meet the performance challenges posed by complex queries and high volume databases.

Acknowledgements

I would like to thank Jim Gray, Zia Mohamed, Jim Murray, and Jim Finnerty for their valuable comments on this paper.

REFERENCES

[Ant91A] G. Antoshenkov. "Optimization in Rdb /VMS," *DEC Professional Magazine by Professional Press, Inc.*, Vol. 10, No. 5, (May 1991).

[Ant91B] G. Antoshenkov. "Dynamic Optimization of a Single Table Access," *Technical Report DBS-TR-5, DEC-TR-765, DEC Data Base Systems Group* (June 1991).

[Ant92] G. Antoshenkov. "Random Sampling from Pseudo-Ranked B+ Trees" *Proceedings of the 18th VLDB conference*, (August 1992).

[Babb79] E. Babb. "Implementing a Relational Database by Means of Specialized Hardware," *ACM Transactions on Database Systems*, Vol. 4, No. 1, (March 1979).

[HoEn91] L. Hobbs and K. England. *"Rdb/VMS: A Comprehensive Guide,"* Digital Equipment Corporation, Chapter 6 (1991).

[GrWa89] G. Graefe and K. Ward. "Dynamic Query Execution Plans," *Proceedings of the ACM SIGMOD Conference*, (May 1989).

[INSS92] Y. Ioannidis, R. Ng, K. Shim, T. Sellis. "Parametric Query Optimization," *Proceedings of the 18th VLDB conference*, (August 1992).

[IoCh91] Y. Ioannidis and S. Christodoulakis. "On the Propagation of Errors in the Size of Join Results," *Proceedings of the ACM SIGMOD Conference*, (June 1991).

[MoHa90] C. Mohan, D. Haderle, Y. Wang, J. Cheng. "Single Table Access Using Multiple Indexes: Optimization, Execution, and Concurrency Control Techniques," *Advances in Database Technology - EDBT'90*, (March 1990).

[OlRo89] F. Olken and D. Rotem. "Random Sampling from B+ Trees," *Proceedings of the 15th VLDB conference*, (1989).

[SACL79] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, T. Price. "Access Path Selection in a Relational Database Management System," *Proceedings of the ACM SIGMOD Conference*, Boston, (June 1979).

[Zipf49] G.K. Zipf *Human Behavior and the Principle of Least Effort*, Addison-Wesley, Reading, MA, (1949)