Processing Aggregate Relational Queries with Hard Time Constraints†

*Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeo K. Taneja**

Department of Computer Engineering and Science
Case Western Reserve University
Cleveland, Ohio 44106

ABSTRACT

We consider those database environments in which queries have strict timing constraints, and develop a time-constrained query evaluation methodology. For aggregate relational algebra queries, we describe a time constrained query evaluation algorithm. The algorithm, which is implemented in our prototype DBMS, iteratively samples from input relations, and evaluates the associated estimators developed in our previous work, until a stopping criterion (e.g., a time quota or a desired error range) is satisfied.

To determine sample sizes at each stage of the iteration (so that the time quota will not be overspent) we need to have (a) accurate sample selectivity estimations of the RA operators in the query, (b) precise time cost formulas, and (c) good time-control strategies. To estimate the sample selectivities of RA operators, we use a run-time sample selectivity estimation and improvement approach which is flexible. For query time estimations, we use time-cost formulas which are adaptive and precise. To use the time quota efficiently, we propose statistical and heuristic time-control strategies to control the risk of overspending the time quota. Preliminary evaluation of the implemented prototype is also presented.

1. Introduction

This paper discusses the problem of processing an aggregate relational algebra (RA) query within a given time quota. That is, we present a methodology to process the query "Evaluate $f(E)$ within T time units" where f is an aggregate function and E is an arbitrary RA expression. This paper restricts f to $COUNT$.

Our approach is as follows. Instead of evaluating $f(E)$, we evaluate its statistical estimator $\hat{f}(E)$ by sampling from operand relations in E. Given the time quota T, we decide about the sizes of the samples, obtain the samples, and evaluate the estimator $\hat{f}(E)$. If, at the end of the evaluation of $\hat{f}(E)$, there is time left then we perform a second stage of obtaining additional samples and improving the estimate for $f(E)$. We continue improving the estimate by additional stages until the time quota is completely used.

Our approach can be used directly in single-user databases to strictly control the database query evaluation times. The time constraint can be set to milliseconds/seconds (e.g., realtime databases) or to minutes (e.g., an interactive environment with an "impatient" user). We are presently using the approach of this paper to build a database system for programmable logic controllers [OzHO 88].

† This research is supported by the National Science Foundation under Grants DCR-860554, and IRI-8811057.

* Department of Mathematics and Statistics, Case Western Reserve University.

Another use of our approach is in multiuser, realtime databases. By precisely fixing the execution times of database queries in a transaction, accurate estimates for transaction execution times becomes possible. This in turn plays an important role in minimizing the number of transactions that miss their deadlines [AbMo 88].

In [HoOT 88], we have developed a general methodology to obtain consistent and unbiased estimators for the aggregate query $COUNT(E)$, where E is an arbitrary relational algebra (RA) expression. No a priori knowledge of the distributions of attribute values is needed in estimating $COUNT(E)$. The basic theoretical framework developed is based on the simple random sampling method, and then extended to a cluster sampling plan for efficiency considerations. Goodman's estimator [Good 49] is revised as an estimator for those expressions that contain the projection operation. The performance of the estimators are reported in [HoOT 88, HouO 88].

In this paper, we discuss, for a given $COUNT(E)$ query and a given time quota, when to stop processing the query (i.e., the *stopping criteria*) to meet the time constraint, and how to use the time quota efficiently (i.e., the *time control strategies*). The time-control algorithm determines the sample size from each operand relation such that a query with the set of sample tuples as input can be evaluated with a desired probability within the given amount of time quota. Clearly, a good time control algorithm needs a precise time-cost formula to determine the sample size, and also needs a good time-control strategy to control the risk of overspending the time quota.

The time-cost formula of an estimator for a $COUNT(E)$ query is taken to be a function of the sample size and selectivities of operators in E obtained by using the samples (i.e., the *sample selectivity*). In this paper, we propose a run-time approach to estimate the sample selectivities of RA operators in a $COUNT$ query because of its flexibility. By taking advantage of the run-time, stage-by-stage evaluation of a query, adaptive time cost formulas are proposed. Statistical and heuristic time control strategies are introduced, and compared in terms of the efficient utilization of the time quota and the control of the risk of overspending the time quota.

The remainder of the paper is organized as follows. In Section 2, a brief review of our previous work [HoOT 88] is presented. In Section 3, we propose the run-time estimation technique to estimate the sample selectivities of RA operators. Also, we discuss and compare the statistical and heuristic time control strategies. In section 4, we introduce the idea of using adaptive time-cost formulas to derive more flexible time-cost formulas. We have implemented the time-control algorithm of section 3 and the associated time-cost formula evaluations in a prototype DBMS. Section 5 gives the preliminary experimental results of the proposed time-control algorithm obtained from the implemented prototype DBMS.

2. Terminology and Previous Work

First, we briefly describe some statistics terminology. Let ψ be a parameter of interest, such as a population mean or a population total. An *estimator* of ψ, denoted by $\hat{\psi}$, is a function that returns a value, based on the observation on the sampled data, serving as a guess for the value of ψ. An estimator $\hat{\psi}$ is said to be *unbiased* if

$\Xi(\hat{\psi}) = \psi$ for all possible values of ψ, where $\Xi(\hat{\psi})$ denotes # the expected value of $\hat{\psi}$. We will also use the notation μ_ψ for $E(\psi)$. The *bias* of an estimator $\hat{\psi}$ is defined to be $\Xi(\hat{\psi}) - \psi$. An estimator is *consistent* if the probability that it is in error by more than any given amount approaches zero as the sample becomes large. Both *unbiasness* and *consistency* are desirable properties of an estimator. The variance of a random variable X, denoted by $Var(X)$, is defined to be $\Xi\{(X - \mu_X)^2\}$, where μ_X is the expected value of the random variable X. The variance is a measure of the variations among the possible values of a random variable. *Confidence interval* and *confidence level* are used to describe how close an estimate is to the true value of a parameter. Confidence interval is an interval of plausible values for the parameter being estimated. Confidence interval is the degree of plausibility of such an interval.

Simple random sampling is a method of selecting m elements (sample size) out of N (population size) such that each one of possible samples that contain m elements has an equal chance of being selected. Since a unit (i.e., an element) that is already selected is removed from the population for all subsequent draws, this method is also called the *random sampling without replacement*. *Cluster sampling (without replacement)* denotes the selection method in which a cluster of elements are drawn as a single sample unit.

In [HoOT 88], we have proposed to use samples of relations to compute an estimate for $COUNT(E)$, where E is an arbitrary RA expression. We first transform $COUNT(E)$ into $\sum_i COUNT(E_i')$ using *the Principle of Inclusion and Exclusion* [Liu 68], where E_i' is an RA expression containing only Select, Join, Intersect and Project operations. Then, we give an estimator for each $COUNT(E_i')$, and, thus, an estimator of $COUNT(E)$ is constructed.

A relation instance r with $|r|$ tuples is modeled as a set of $|r|$ points in a one-dimensional space. A Select-Join-Intersect-Project expression E with n operand relations $r_1, r_2, ..., r_n$ is modeled as an n-dimensional space, called *the point space of E*, such that

(i) Each relation corresponds to one dimension : There are totally $\prod_{i=1}^{i=n} |r_i|$ points in the point space, where $|r_i|$ is the number of tuples in the relation r_i, assuming that the n different operand relations are numbered from 1 to n. Each point uniquely maps to a sequence of n-tuples $(t_1, ..., t_n)$, where $t_i \in r_i$, $1 \le i \le n$. Figure 2.1. shows the mapping between a point and the corresponding 2-tuples (t_1, t_2) where $t_1 \in r_1$, $t_2 \in r_2$.

(ii) A point in the point space, assumes the value 1 if the corresponding $(t_1, ..., t_n)$, when substituted into E, produces an output tuple; otherwise the point has the value 0.

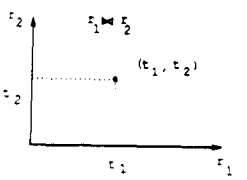


Figure 2.1. Mapping Between a Point and Tuples

For a Select-Join-Intersect expression E, computing $COUNT(E)$ is equivalent to counting the number of points with the value 1 in the point space. In [HoOT 88], a consistent and unbiased estimator, denoted by $\hat{Y}(E)$, is proposed for estimating the number of 1's in the point space. $\hat{Y}(E)$ is defined as $N.(y / m)$, where N is the total number of points in the point space of E, m is the number of sample points, and y is the number of sample points with the value of 1. The formulas for computing an unbiased estimate of the variance of $\hat{Y}(E)$ are also given in [HoOT 88].

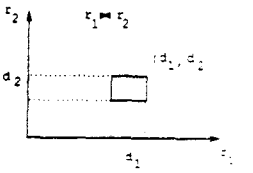


Figure 2.2. Mapping Between a Space Block and Disk Blocks

A set of tuples which have the same values for the projected attributes after the projection operation, become a single tuple. For a Select-Join-Intersect-Project expression E, computing $COUNT(E)$ is equivalent to counting the number of different *groups* of n-tuples such that each group maps to a unique 1 value in the point space. Goodman's estimator, based on the occupancies of groups in the sample, is proposed in [HoOT 88] for estimating $COUNT(E)$.

In [HoOT 88], we first develop the $COUNT(E)$ estimation methodology based on the simple random sampling, and then extend it to a new sampling plan which is based on the cluster sampling technique. In the cluster sampling plan, a disk block is taken as a sample unit (i.e., all the tuples in a disk block are taken as a whole) from each operand relation. As a result, the point space can be viewed to be divided into $\prod_{i=1}^{i=n} D_i$ *space blocks*, where D_i is the number of disk blocks in relation r_i, and each space block uniquely maps to a sequence of n disk blocks $(d_1, ..., d_n)$, where d_i is a disk block in relation r_i, $1 \le i \le n$. Figure 2.2 pictures the mapping between the space blocks and disk blocks.

In [HoOT 88], an estimator based on the cluster sampling plan, denoted by $\hat{Y}_b(E)$, where b emphasizes that disk blocks are sample units, is proposed for a Select-Join-Intersect expression E. $\hat{Y}_b(E)$ is defined as $B(\sum_{i=1}^{i=b} y_i / b)$, where B is the total number of space blocks in the point space of E, b is the number of space blocks in the sample, and y_i is the number of points with the value 1 in the i_{th} space block, assuming that the set of sample space blocks are numbered from 1 to b. A formula for computing the variance of $\hat{Y}_b(E)$ can also be found in [HoOT 88]. In the cluster sampling plan, disk blocks are randomly chosen from each operand relation and evaluated almost the same as in the ordinary query evaluation, except that union and difference operations are replaced by the intersection operation using the Principle of Inclusion and Exclusion in the $COUNT(E)$ evaluation. The cluster sampling plan has the advantages of efficiency in sampling and in evaluation.

3. Time-Constrained Aggregate Relational Query Evaluation Algorithm

The query evaluation algorithm is given in Figure 3.1. Essentially, the algorithm repetitively gets a set of sample disk blocks and evaluates the estimator until the stopping criterion is satisfied. Each iteration of the *while-loop* is called a *stage*, and includes the steps of determining the sample size, retrieving and evaluating the sample tuples, and computing an estimate of *COUNT(E)*.

We use the notation Ξ for expected value (rather than its standard notation E) to eliminate any confusion with the RA expression E.

Algorithm Time-Constrained-Aggregate-Relational-Query-Evaluation (E, T)

Input : E : an arbitrary RA expression.
T : a given amount of clock time (quota).
Output : an estimate of $COUNT(E)$ given out within T clock time units.
Variables : i : the number of the current stage; $i = 1$ initially.
T_i : the amount of clock time left after $i-1$ stages; $T_1 = T$.
f_i : the sample fraction to be taken at the i_{th} stage.
SEL_1^{i-1} : the set of sample selectivities of operators in E obtained from the samples of the previous $i-1$ stages.
START-TIME : the starting time of the query evaluation.
CURRENT-TIME : the current time.
SAMPLE-SET : a set of random numbers, denoting the disk blocks retrieved up to and including the $(i-1)_{th}$ stage.
NEW-SAMPLE-SET : a set of random numbers, denoting the sample disk blocks drawn at the present stage.
Stopping-Criterion : a boolean variable whose value determines whether the execution should terminate or not. When TRUE, execution terminates. Its value is initially FALSE, and is set to TRUE by the associated timer interrupt service routine.
Procedures used:
Revise-Selectivities(E, i) : Computes the set of sample selectivities of operators, i.e., SEL_1^{i-1}, based on the samples of the previous $i-1$ stages.
Sample-Size-Determine(E, i, T_i, SEL_1^{i-1}) : determines the sample fraction to be used at the i_{th} stage.
New-Sample-Select(f_i) : returns the set of random numbers, denoting the disk blocks to be accessed at the present stage i with the sample fraction f_i.

```
begin
    Stopping-Criterion := FALSE;
    i := 1;
    T_i := T;
    SAMPLE-SET := φ;
    Read the current clock time into START-TIME;
    Set the timer interrupt to T units;
    while Stopping-Criterion = FALSE do begin
        SEL_1^{i-1} := Revise-Selectivities(E, i);
        f_i := Sample-Size-Determine (E, i, T_i, SEL_1^{i-1});
        NEW-SAMPLE-SET := New-Sample-Select(f_i);
        Compute an "improved" COUNT(E);
            { by reading and evaluating the samples specified in
              NEW-SAMPLE-SET, and recomputing COUNT(E) }
            { COUNT(E) evaluation may or may not use the
              SAMPLE-SET, depending on the implementation }
        SAMPLE-SET := SAMPLE-SET ∪ NEW-SAMPLE-SET;
        i := i + 1;
        Read the current clock time into CURRENT-TIME;
        T_i := T - (CURRENT-TIME - START-TIME);
    endwhile;
    return( COUNT(E) );
end.
```

Figure 3.1. Time Constrained Query Evaluation Algorithm

In Figure 3.2, we summarize the prototype DBMS implementation decisions we have made for our time-constrained *COUNT* query evaluation, and discuss them below. A more detailed discussion is in [HoOT 88a]. We discuss only *COUNT(E)* type queries in this paper. However, most of the following discussions apply to any type of relational algebra query (given, of course, an estimator for the query).

3.1 Estimating the Selectivities of Operators

We define the *selectivity* of a Select-Join-Intersect-Project expression E to be the ratio of the number of output tuples (i.e., the number of points with the value 1 or the number of different groups of points with the value 1 if E contains projection operations) to the product of the number of tuples in input relations (i.e., the total number of points in the point space of E). Since the union and difference operations in $COUNT(E)$ evaluation are replaced by the intersection operation by applying the principle of inclusion and exclusion, we don't need to compute the selectivities of the union and difference operations.

A time-cost formula for $COUNT(E)$ estimator evaluation is a function of the sample size and the selectivities of RA operators in E obtained by using the sample (*not* the input relations).

Sample size can be stated in absolute measures (e.g., the number of tuples or the number of disk blocks) or in a relative measure (e.g., the sample fraction f), and these measures can transform to each other easily. Sample fraction f is defined as $f = d/D \approx m/N$, where d is the number of sample disk blocks from the relation, D is the total number of disk blocks in the relation, m is the number of tuples in the sample, and N is the total number of tuples in the relation. For the simplicity of the discussion and the ease of solving the time-cost formula for sample size, we assume that *equal sample fractions are taken from all of the relations*. Also, one advantage of using equal sample fractions for sample sizes is that the sample space blocks spread evenly in the point space along each dimension for a relation, and, hence, estimates with probably higher precision may be obtained.

Sample selectivity of each operator Op in $COUNT(E)$ may be approximated by the selectivity of Op obtained by using the input relations $r_1, r_2, ..., r_n$. In such a case, the time-cost formula of the estimator for COUNT(E) can be used to solve for the only unknown left, i.e., the sample size. There are two approaches for determining the selectivities of operators :

(1) use the distribution functions of relations to derive selectivities of operators [Chri 83, SACL 77, etc.], or
(2) use prestored statistics to estimate the selectivities [PSCo 84, Rowe 85, MuDe 88, etc.].

Deriving from the distributions of input relations a closed form distribution function for the output relation is an open problem, and is not very promising.

Prestored selectivities of operations in a query [PSCo 84, Rowe 85, MuDe 88, etc.] can be obtained by preevaluating (partially or completely) the query with input relations. This approach is simple and may have a very good performance. However, an extra effort is needed to maintain the set of stored selectivities when there are changes to the database. In order to suit for general database use, for each relational algebra operation, a large set of selectivities which are associated with all the possible combinations of attributes, input relations, selection formulas and projected attributes, have to be stored. This approach is best suited for database environments where only a fixed set of query types are to be issued, and we will not pursue it here.

The approach we use in this paper is to directly estimate and improve sample selectivities at each stage. We call this *the run-time estimation approach*. That is, the selectivity of an operation is estimated at run-time, and also the precision of the estimated sample selectivity is improved at run-time (see Figure 3.3). The run-time estimation approach has the greatest flexibility because it does not need any specific information about a query. For the first stage, we assume a reasonably large selectivity for each operation, and determine the size of the first-stage sample. The first stage sample is

	Choices	Implementation Decision	Advantages
Selectivity Estimation	• Attribute Distributions • Prestored Statistics • Run-Time Estimation	Run-Time Estimation	Flexible and No a priori Knowledge Needed
Stopping Criteria	• Hard Time Constraint • Soft Time Constraint	Hard Time Constraint	Satisfying the time constraint is the goal
Time-Control Strategies	• Statistical Strategies • Heuristic Strategy	One-at-a-Time-Interval Statistical Strategy	Simple and Controlled Risk
Sampling Techniques	• Simple Random Sampling • Cluster Sampling	Cluster Sampling with Full Fulfillment	Time Efficient
Time Cost Formulas	• Fixed Form • Adaptive	Adaptive	Precise and Flexible
Maintenance of Sample Disk Blocks	• In Main Memory • In Secondary Storage	In Secondary Storage	Storage Efficient (but not time)

Figure 3.2. Table of Implementation Decisions in Our Prototype DBMS

used for obtaining a better selectivity estimation as well as for evaluating the estimator for the query. After evaluating the estimator using the first-stage sample, if there is additional time left to improve the estimator, a second-stage sample is drawn using the previously estimated sample selectivities and so on. This iterative process continues until the time quota is consumed.

3.2. Stopping Criterion

We now consider the issue of determining when the system should stop processing a query and give out the result. Basically, there are two types of stopping criteria. The first type of criteria is concerned about the constraint of time while the other type of criteria is concerned about the precision of estimation. The simplest criterion of the first type is to stop immediately, upon receiving a timer interrupt indicating the time quota has been spent and give whatever has been obtained through the last stage as the result. In such a case, the current stage of execution, which is not yet finished, has to be aborted immediately and the amount of time spent in the last stage is simply wasted. Clearly, this stopping criterion implies a need for good time-control strategies such that the amount of time wasted can be reduced. This criterion is also referred to as the *hard time constraint* or *hard deadline* in the real-time database context [AbGM 88]. Variations of this criterion are possible. For example, by defining a *value function* for the completion time of a query, the system decides when to stop processing the query to get a higher value. Criteria of this type are referred to as *soft time constraints* or *soft deadlines* [AbGM 88]. In a hard time constraint environment, the execution is interrupted whenever the time quota is consumed, instead of waiting for the end of the *while-loop* in Figure 3.1 to check the status. In our implementation, we have chosen to use the hard time constraint as the stopping criterion because of its simplicity and wide applicability in the real-time database environment. However, the algorithm shown in Figure 3.1 actually implements a soft time constraint (for simplicity in the presentation).

The second type of criteria is to stop whenever the precision of estimation has met the user's requirement or whenever the estimation does not improve "much" over the last few stages even though there may still be some time left. In either case, continuing the processing of a query is considered to be unnecessary. This criterion plays an important role in another research problem of *error-constrained* query evaluation, which will not be discussed here. Combinations of both types of criteria are also possible.

Procedure Revise-Selectivities (E, i)

Input : E : an arbitrary RA expression.
i : the number of the current stage.

Output : SEL_1^{i-1} : the set of sample selectivities of operators in E obtained from the samples of stages 1 through $i-1$.

Variables : Op : an RA operator in E.
$sel_1^{i-1}(Op)$: the sample selectivity associated with the operator Op obtained from the samples of stages 1 through $i-1$; $sel_1^0(Op)$ is the value in the first stage.
r_i : the operand relations of Op, $1 \le i \le 2$.
$tuples_j(Op)$: the number of output tuples of the operator Op at the j_{th} stage, $1 \le j \le i-1$.
$points_j(Op)$: the number of sampled points in the point space of Op at the j_{th} stage, $1 \le j \le i-1$.

Procedures : $maximum(|r_1|, |r_2|)$: returns the maximum number of tuples in relation r_1 and r_2.

```
begin
  For each operator Op in E do
    If i = 1 then    { assign the maximum selectivity at the first stage }
      If Op ∈ { Select, Project, Join }
        sel_1^0(Op) := 1;
      else                          { Op is Intersection }
        sel_1^0(Op) := 1 / maximum(|r_1|, |r_2|);
    else
      sel_1^{i-1}(Op) := Σ_{j=1}^{j=i-1} tuples_j(Op) / Σ_{j=1}^{j=i-1} points_j(Op);
end
```

Figure 3.3 Initialization and Improvement of the Selectivity Estimations for the RA Operators in E

3.3. Time Control Strategies

A time-control algorithm not only has to make the query processing meet the time constraint, but also, for a given amount of time quota, it should produce an estimate as precise as possible. Normally, the larger the sample size is, the more precise the estimate will be. That is, for a given amount of time, the goal is to evaluate as large a sample as possible. In the run-time approach, the more stages it goes through, the more overhead is involved. Moreover, for operations such as intersection and join, besides the increase in overhead, as the number of stages increases, the time complexity increases for a given overall sample size [HoOT 88a]. This implies

that, at each stage, the strategy should allocate as large an amount of time as possible to reduce the number of stages. On the other hand, allocating large amounts of time has the risk of overspending the time quota and may end up wasting a large amount of time in a hard time-constrained environment. Therefore, a tradeoff has to be made between the number of stages (i.e, the overhead) and the amount of time wasted (i.e., the risk of overspending). Below, there are three possible approaches, the *Single-Interval strategy* and *One-at-a-Time-Interval strategy*, which are categorized as *statistical strategies*, and the *heuristic strategy*. We do not discuss the heuristic strategy here.

3.3.1 Single-Interval Strategy

Assume that we are at the i_{th} stage. Let T_i be the amount of time left after $i-1$ stages, and t_i be the random variable representing the actual amount of time that will be spent at the i_{th} stage with mean μ_{t_i} and variance $\mathrm{Var}(t_i)$. Let f_i be the sample fraction. $sel_i^i(Op)$ (or, simply, sel_i^i) denotes the sample selectivity of the operator Op at the i_{th} stage and $sel_1^k(Op)$ (or sel_1^k) denotes the estimated sample selectivity of the operator Op obtained from all the sample results, from stages 1 to stage k combined. $SEL_i^i(E)$ (or, simply, SEL_i^i) denotes the set of $sel_i^i(Op)$ for each operator Op in E (i.e., $sel_i^i \in SEL_i^i$). The *risk of overspending at stage i*, denoted by α_i, is defined to be $P(t_i > T_i)$, where P denotes the probability.

The *statistical strategies*, which refer to the Single-Interval and One-at-a-Time-Interval [JoWi 82] strategies, are based on the concepts of *confidence interval and level*, and estimate the amount of time to spend at each stage and, at the same time, keep the risk in control. The equality

$$t_i = QCOST(f_i, SEL_i^i)$$

is satisfied, where $QCOST(f_i, SEL_i^i)$ is the time-cost formula of the query at the i_{th} stage. Note that SEL_i^i and, hence, t_i are unknown until the stage i ends. Therefore, we use the expected version of the above equation, i.e.,

$$\mu_{t_i} = \Xi\left\{ QCOST(f_i, SEL_i^i) \right\} \qquad (3.1)$$

Now in order to solve the equation (3.1) for f_i, we need to find μ_{t_i}, which is done by controlling the risk as follows. For a given risk α_i, a number d_{α_i} can be obtained such that the actual amount of time spent at stage i will be less than or equal to $\mu_{t_i} + d_{\alpha_i}\sqrt{Var(t_i)}$ with probability $1-\alpha_i$, i.e., $P(t_i \le \mu_{t_i} + d_{\alpha_i}\sqrt{Var(t_i)}) = 1-\alpha_i$. Therefore, if we let

$$T_i = \mu_{t_i} + d_{\alpha_i}\sqrt{Var(t_i)} \qquad (3.2)$$

then t_i will be less than or equal to T_i with probability $1-\alpha_i$. To summarize, μ_{t_i} is obtained by solving the equation (3.2), and then f_i is obtained by solving the equation (3.1).

Note that, $\Xi\{QCOST(f_i, SEL_i^i)\}$ is very costly to compute since sel_i^i in SEL_i^i are not independent, and covariances between sel_i^i's need to be computed.

At each stage i, one can use SEL_1^{i-1} (known from the samples of the previous $i-1$ stages) as a plausible value for SEL_i^i. Similarly, covariances between sel_1^{i-1}'s (which can be computed from the samples of previous stages) can be used as plausible values for covariances between sel_i^i's in the computation of the $\Xi\left\{ QCOST(f_i, SEL_i^i) \right\}$. And, for the variance $\sqrt{Var(t_i)}$, which is a function of the sample fraction f_i and the variation of the values among sample units (space blocks) at stage i, one can use the variation among all the previously sampled units as a plausible value of the variation among the i_{th} sample units.

Different values for α_i can be chosen at different stages. A possible choice is to adjust the values based on the amount (or percentage) of time left. The idea is that when there is a small amount of time left, instead of loosing time in overhead, it may be reasonable to take a higher risk of allocating a larger amount of time.

3.3.2 One-at-a-Time-Interval Strategy

An alternative approach in statistics category is to apply the concept of *one-at-a-time intervals* [JoWi 82]. At each stage i, instead of considering the risk of overspending in the whole RA query expression, the risk of overspending in each operation, denoted by β_i, is considered individually. That is, at stage i, we use $sel_i^{i+}(Op)$ (or, simply, sel_i^{i+}) for Op such that $sel_i^{i+} \ge sel_i^i$ with probability $1-\beta_i$, i.e., $P(sel_i^{i+} \ge sel_i^i) = 1-\beta_i$. Such a selectivity sel_i^{i+} can be derived by using the equation

$$sel_i^{i+} = \mu_{sel_i^i} + d_{\beta_i}\sqrt{Var(sel_i^i)} \qquad (3.3)$$

where $\mu_{sel_i^i}$ is the mean of sel_i^i, $Var(sel_i^i)$ is the variance of sel_i^i, and d_{β_i} is a proper value chosen by the system (based on the distribution of sel_i^i) for controlling the risk β_i.

In equation (3.3), we may use sel_1^{i-1} (which is known from the samples of previous stages) as a plausible value for $\mu_{sel_i^i}$. And, for the variance $Var(sel_i^i)$, which is a function of the sample size (fraction) f_i and the variation of the values among sampled units (space blocks) at the i_{th} stage, one can use the variation among previously sampled units as a plausible value of the variation among the i_{th} stage sample units. We discuss the approximation of $Var(sel_i^i)$ later in this section.

We want to spend as much time at stage i as possible i.e.,

$$T_i = QCOST(f_i, SEL_i^{i+}) \qquad (3.4)$$

where SEL_i^{i+} denotes the set of sel_i^{i+}'s.

To summarize, for each Op in E of $COUNT(E)$, we use the equation (3.3) to obtain $sel_i^{i+}(Op)$. The property $P(sel_i^{i+} \ge sel_i^i) = 1-\beta_i$ is satisfied by sel_i^{i+}. We then use the equation (3.4) to obtain f_i.

A comparison of the two approaches of considering the confidence level of the whole query and one-at-a-time for each operator can be made as follows. The first approach reserves a certain amount of time, i.e., $d_{\alpha_i}\sqrt{Var(t_i)}$, to prepare for the extra amount of time to spend, due to possible underestimation of the selectivities during the processing of the query at stage i. On the other hand, the second approach assumes a reasonably high selectivity for each operation, i.e., sel_i^{i+} (without reserving any time) to achieve the same goal of not overspending. The first approach considers the query as a whole while the second approach considers each operation individually without taking into account the relationships among the operations. Therefore, the first approach may have a better control of the overall risk in query processing, but requires more effort in computing the expected time cost of the query given by $\Xi\{QCOST(f_i, SEL_i^i)\}$ (which includes computing the expected values of each term of the time-cost formula) and in computing the variance of the query cost $\sqrt{Var(t_i)}$ (which includes computing the variances of each term and covariances of each pair of terms in the cost formula), and this is a very expensive procedure. As for the second approach, since it does not consider the relationships among operations, much less computation (only the variance of the selec-

tivity of each operation $Var(sel_i^i)$), is needed. However, in the second approach, we do not know what to assert about the overall probability of finishing the processing of the sample at each stage. We believe that, with experience, an algorithm based on the One-at-a-Time-Interval approach can be more efficient than the Single-Interval, because of its simplicity. We have chosen to use the One-at-a-Time-Interval approach as the basis of the time-control algorithm in our implementation.

We now turn our attention to the approximation of $Var(sel_i^i)$ in equation (3.3). Let us make the assumption that the variance of the sample selectivity sel_i^i is equal to the variance of the same selectivity obtained by using the input relations. Then, one can directly use the variance formula given in Theorem 6 of [HoOT 88] for the variance computation of the selectivity of an operation. For the cluster sampling plan, the computation of $Var(sel_i^i)$ needs first to sort the output tuples, according to disk numbers from which their corresponding source tuples come, to determine the value of a space block and then compute the variance using the given formula. Unfortunately, the above two processes, i.e., sorting and computation of the formula, are too expensive. Therefore, in our implementation we have chosen to use the variance formula for simple random sampling (without replacement of points) as an approximation to $Var(sel_i^i)$, which is explained below.

Consider an RA expression E_1. Let S be the proportion of the points in the point space of E_1 with the value 1, i.e., S is the selectivity of E_1 with respect to the input relations. Let s be the sample selectivity of E_1. Then, it is known in statistics (e.g., see [Coch 77]) that for a simple random sample of m points from E_1's point space of N points, $Var(s) = S\ (1-S)(N-m)\ /\ (\ m\ (N-1))$.

At the i_{th} stage, a new sample of size m_i is drawn from the set of points (of size N_i) which have not been included in the samples of previous $i-1$ stages. sel_1^{i-1} can be used as a plausible value for the expected value of sel_i^i. Therefore, the variance of sample selectivity at stage i based on a simple random sampling of points can be estimated as

$$Var(sel_i^i) = \frac{sel_1^{i-1}\ (1 - sel_1^{i-1})(N_i - m_i)}{m_i\ (N_i - 1)}$$

We have used the above equation as an approximation for the variance formula of the cluster sampling plan. Usually, the approximation gives a smaller value for $Var(sel_i^i)$. Therefore, some inaccuracy in the risk control is expected. However, it greatly reduces the amount of time spent on the computation of variance. We will see how this approximation will affect the experimental results in Section 5.

Figure 3.4 shows one possible implementation of the One-at-a-Time-Interval approach, which is used in our prototype implementation. The cost of the query, *QCOST*, is the sum of the costs of all the operators. *QCOST* (obtained during the last iteration of the *for-loop* in the algorithm) is the operator cost of the outermost operator Op in E of COUNT(E). The variables *low* and *high* will, at each step, be replaced by the highest f_i value such that $\mu_{t_i} < T_i$ and the lowest f_i value such that $\mu_{t_i} > T_i$, respectively. The f_i value that satisfies equations (3.3) and (3.4) is returned by the algorithm Sample-Size-Determine.

The computation of sel_i^{i+} of an operation Op (i.e., equation (3.3)) is given in the procedure $ComputeSel_i^{i+}\ (Op, f_i, sel_1^{i-1})$, where $Op \in$ { Select, Join, Union, Difference, Intersect, Project }. Figure 3.5 shows the implementation of the procedure $ComputeSel_i^{i+}\ (Op, f_i, sel_1^{i-1})$, which uses the variance formula for simple random sampling as an approximation for the computation of $Var(sel_1^i)$.

Algorithm Sample-Size-Determine (E, i, T_i, SEL_1^{i-1})

Input : E : an RA expression.
i : the number of the current stage.
T_i : the amount of time left for the i_{th} stage.
SEL_1^{i-1} : selectivities of operators obtained from the samples of previous $i-1$ stages.
Output : a sample fraction f_i to be taken at the i_{th} stage.
Variables: Op : an RA operator.
f_i, *low*, *high* : sample fractions, $0 \le low \le f_i \le high \le 1$.
sel_i^{i+} : the selectivity of the operator Op at the i_{th} stage.
μ_{t_i} : the expected amount of time to be spent at the i_{th} stage with each operator assumed the selectivity sel_i^{i+}.
ε : a system-defined constant denoting the tolerable error in choosing a μ_{t_i} as close to T_i as possible.
r_i : a base or an intermediate relation; r_1 and r_2 denote the operand relations, and r_0 denotes the output relation of the RA operator Op.
Procedures used :
$OpCost\ (r_0, r_1, r_2, f_i, sel_i^{i+})$: returns the time cost of the RA operator Op that generates the output relation r_0 using the samples taken from r_1 and r_2 in accordance with the sample fraction f_i.
$ComputeSel_i^{i+}\ (Op, f_i, sel_1^{i-1})$: returns sel_i^{i+} of the operator Op.

```
begin
  Choose a value for d_{β_i};
  f_i := low := μ_{t_i} := 0;
  high := 1;
  while | μ_{t_i} - T_i | > ε do begin
    if μ_{t_i} < T_i then low := f_i else high := f_i;
    f_i := (low + high) / 2;
    μ_{t_i} := 0;
    for each Op in E do begin        { in execution order of Op's }
      sel_i^{i+} := ComputeSel_i^{i+} (Op, f_i, sel_1^{i-1});
      switch Op
      case Select    : μ_{t_i} := μ_{t_i} + SelectCost(r_0, r_1, f_i, sel_i^{i+});
               break;
      case Join      : μ_{t_i} := μ_{t_i} + JoinCost(r_0, r_1, r_2, f_i, sel_i^{i+});
               break;
      case Union     : μ_{t_i} := μ_{t_i} + UnionCost(r_0, r_1, r_2, f_i, sel_i^{i+});
               break;
      case Difference : μ_{t_i} := μ_{t_i} + DifferenceCost(r_0, r_1, r_2, f_i, sel_i^{i+});
               break;
      case Intersect : μ_{t_i} := μ_{t_i} + IntersectCost(r_0, r_1, r_2, f_i, sel_i^{i+});
               break;
      case Projection : μ_{t_i} := μ_{t_i} + ProjectionCost(r_0, r_1, f_i, sel_i^{i+});
    end
  endwhile;
  return(f_i);
end.
```

Figure 3.4. Sample Size Determination Algorithm

3.4 A Stage With a Zero Estimated Selectivity for an Operation

As mentioned earlier, the run-time estimation algorithm is intended for general time-constrained database applications. As a result, it is quite possible that at the first stage (or even the first few

Procedure $ComputeSel_i^{i+}(Op, f_i, sel_1^{i-1})$

Input : Op : a relational algebra operator in E.

f_i : the sample fraction to be taken at the i_{th} stage.

sel_1^{i-1} : selectivity of the operator Op obtained from the samples of previous $i-1$ stages.

Output : sel_i^{i+} : an (assumed) selectivity for the operator Op, $sel_i^{i+} \geq sel_1^{i-1}$.

Variables : β_i : a risk (of overspending) that the system is willing to take at the i_{th} stage.

d_{β_i} : a value chosen such that the computed sel_i^{i+} will have the probability $1-\beta_i$ of being larger than sel_1^{i-1}.

m_j : the number of sample points in the point space of Op at the j_{th} stage;

N : the total number of points in the point space of Op.

N_i : the number of points not included in the samples of the previous $i-1$ stages in the point space of Op.

begin

$\mu_{sel_i^i} := sel_1^{i-1}$;

$N_i := N - \sum_{j=1}^{j=i-1} m_j$;

$sel_i^{i+} := \mu_{sel_i^i} + d_{\beta_i}\sqrt{\mu_{sel_i^i}(1-\mu_{sel_i^i})(N_i - m_i) / (m_i (N_i - 1))}$;

end.

Figure 3.5. Computation of sel_i^{i+} values

stages), some of the operators have sample selectivities of zero, due to the small sample sizes. Those operations with sample selectivities of zero will have zero as the value of the estimated variance of the space blocks since all the sampled points have the value zero. This implies that there will be no improvement for the sample selectivity changes (actually larger) for these operations at a later stage. For example, let's consider applying the One-at-a-Time-Interval strategy to the query $COUNT(r_1 \bowtie r_2)$, where r_1 and r_2 are relations. Assuming that after the first stage evaluation, the query has no output tuples, this implies that $sel_1^1 = 0$, and $Var(sel_1^1) = 0$, and, hence, $sel_2^{2+} = 0$ (from equation (3.3)) is too small. Clearly, unless there are no output tuples at the second stage (and for all the latter stages), the quota will be overspent.

Our solution to this situation (implemented in our prototype) is to compute a different selectivity (> 0) for the operation using a combinatorial formula (which is closed and easy to compute). See [HoOT 88a] for details.

4. Estimator Evaluation and Time Cost Formulas

In this section, we give the evaluation algorithms for each RA operator Op in $COUNT(E)$ query as well as the associated time cost formulas for Op (which is used in the algorithm Sample-Size-Determine).

The time control strategies require a precise cost formula for the query. The time cost of a query at a stage is computed as the sum of the cost of relational operations which use samples as their operand relations. The cost of determining the sample sizes (i.e., the time spent in the *while-loop* of Figure 3.5) is very small, and considered as part of the "overhead", which is measured at run-time. In this section, we briefly give the RA operator evaluation algorithms used in the estimator evaluation, and derive the associated cost formulas.

Before deriving the cost formulas, we give an example to show how the sampled disk blocks are evaluated in a stage-by-stage query evaluation environment. Now, let's consider the query COUNT($r_1 \bowtie r_2$) in Figure 4.1. For simplicity, we only discuss samples from the first two stages. In the figure, 1's and 2's denote the disk blocks taken as samples from relations at stages 1 and 2, respectively. Space blocks shaded by '/' and '\' denote blocks evaluated at the first and the second stages, respectively. Let s_{ji} denote the i_{th} stage sample from relation r_j with sample size (i.e., the number of disk blocks) n_{ji}. The ways of evaluations, described in the figure, can be represented as $(s_{11} \cup s_{12}) \bowtie (s_{21} \cup s_{22})$. This implementation is regarded as a *full fulfillment* of the cluster sampling plan in [HoOT 88]. (Another implementation, a *partial fulfillment*, is less costly, and can be found in [HoOT 88a])

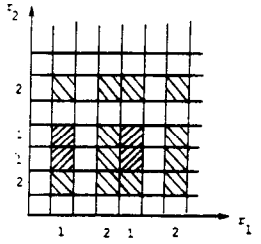


Figure 4.1 Sample Space Blocks to Be Evaluated

In the full fulfillment approach, the number of space blocks evaluated increases by the amount $n_{1s}n_{2s} + (\sum_{i=1}^{i=s} n_{1i})n_{2s} + (\sum_{i=1}^{i=s} n_{2i})n_{1s}$ at the stage s. The full fulfillment approach has the advantage of making the most use of the sampled data, and hence it is time-efficient. The disadvantage is that the intermediate results, from all the previous stages, have to be kept for later processing with new samples at future stages.

In the following sections, we derive the time cost formulas of RA operations implemented in our prototype database management system. This system uses relational algebra expressions as its query language, and has been extended with the new features of cluster sampling, RA operations on samples, and the time control mechanism for time-constrained query processing applications. Different implementations of RA operators for samples may lead to different cost formulas. However, this will not affect the application of the idea of adaptive time cost formulas and the justification of the time control strategies.

We have made a design decision that all the input relations and all the intermediate relations are always kept on disks. This decision is mainly motivated by our assumption that, in very large databases, even samples may be too large to fit into the main memory. One can also revise the time-cost formulas of this section to give a "main-memory-only" evaluation of samples. A main-memory-only version of the prototype DBMS is also being developed now, and will incorporate a version such that, after samples are taken, all data processing is confined to the main memory. We believe that when large main memory is available, the sampling approach with a time-control mechanism can be efficiently implemented and will be very promising for real-time database applications.

The time cost formula of an RA operation involving samples is taken to be a function of

(a) sample sizes,
(b) selectivities of operations on the sample tuples, and
(c) a set of parameters (coefficients) which emphasize specific characteristics of a query such as tuple sizes of sample relations, number of join attributes, comparisons in selection formulas, etc..

We think that using a fixed-form cost formula for an operation (i.e., one with all the values of coefficients fixed) is not flexible enough to cope with the differences in the characteristics of sample relations

(e.g., tuple sizes) and characteristics of operations themselves (e.g., different selection formulas). Our approach is to use *adaptive time cost formulas* by taking advantage of the stage-by-stage evaluation in the run-time estimation approach. That is, during run-time, the cost formulas (more specifically, their coefficients) are adjusted based on the sample results to better fit a specific query.

The adaptive approach is as follows. First, we identify the time-consuming steps (e.g., subroutines, procedures) of an RA operation and derive a cost formula for each such step. Then, during the execution of the operation, we record the actual amount of time spent on each step and, based on it, we dynamically adjust the coefficients of the cost functions for each step such that the cost functions can better fit a user's specific query during the latter stages. As for the initialization, the coefficients are assigned initial values that are based on the experimental relations which (designers think) are commonly encountered.

Clearly, the cost formulas also depend on the availability of index files. In what follows, we assume that no index files are used for any RA operation evaluation to simplify the discussion. We use sel^+ to denote sel_i^+ of an operation and f to denote f_i. C_i denotes a constant, and c_j denotes the coefficient whose value is adjusted during the run time. n denotes the number of input tuples to an operation from a relation at the current stage. If the operand relation is a base relation then $n = f \times N$, where f is the sample fraction and N is the total number of tuples in the relation. Otherwise, n is the number of output tuples from the preceding operation. p denotes the number of output pages (of an operation) written to the disk, which can be obtained by the equation $p = sel \times points \;/\; blockingfactor$, where *points* is the number of points in the point space of the relations involved in the operation and *blockingfactor* is the number of tuples in a disk page.

In the following cost formulas, instead of directly specifying the sample fraction f and the selectivities sel's of operations in the time cost formulas, for simplicity, we use n, the number of input tuples in the sample, and p, the number of output pages produced after evaluating the sample input tuples, to express the cost formulas.

4.1 Selection

The algorithm Select in Figure 4.3 describes the estimator evaluation for the selection operation inside a COUNT query. Please note that the estimator evaluation for the selection operation is exactly the same as the regular selection operation evaluation in a relational DBMS. That is, with cluster sampling, standard relational DBMS query processing techniques apply.

Algorithm Select(r_1, F)

```
for  every tuple t in the sample of relation r1  do
     if  t satisfies the qualification F  then  output t;
```

Figure 4.3. Selection Algorithm Used in Estimator Evaluation

The cost of the *for-loop* is $c_1 n + C_1 p$, where c_1 is the cost of reading a tuple from the disk *and* checking a tuple for the satisfaction of the selection formula, and C_1 is the cost of writing a page to the disk.

$$SelectCost(r_0, r_1, f, sel^+) = c_1 n + C_1 p + C_2 \qquad (4.1)$$

where r_0 denotes the output relation, r_1 denotes the input relation and C_1 denotes the constant cost for the initialization task of the operation. As an illustration, we now show that the equation (4.1) can indeed be rewritten as a function of the sample fraction f and the selectivity sel^+ of the select operator (as we have claimed that the time-cost of an RA operator is basically a function of the sample size (fraction) and the selectivity of the operator). That is,

$$SelectCost(r_0, r_1, f, sel^+) = c_1 N f + C_1 N sel^+ + C_2$$

For simplicity, we have assumed that r_1 is a base relation in the above equation. If that is not the case then n, the number of input tuples to the operator, can always be expressed as a function of the sample fraction and selectivities of preceding operators.

4.2 Union and Difference

Let r_1 and r_2 be two degree- and attribute-compatible relations. $COUNT(r_1 \cup r_2)$ and $COUNT(r_1 - r_2)$ are computed as $COUNT(r_1) + COUNT(r_2) - COUNT(r_1 \cap r_2)$ and $COUNT(r_1) - COUNT(r_1 \cap r_2)$ respectively, by the Principle of Inclusion and Exclusion. Therefore, in a $COUNT(r_1 \cup r_2)$ query, $r_1 \cup r_2$ manipulations are replaced by r_1, r_2 and $r_1 \cap r_2$ manipulations, and in a $COUNT(r_1 - r_2)$ query, $r_1 - r_2$ is replaced by r_1 and $r_1 \cap r_2$. As we can see, regardless of the operation being one of union, difference or intersection in the estimator evaluation, only the intersection operation ($r_1 \cap r_2$) needs to be performed. That is, estimator evaluations for union, difference and intersection operations, have exactly the same time cost formula in a $COUNT(E)$ query.

4.3 Intersect

Algorithm Intersect (r_1, r_2)

```
begin
   Write sample tuples from r1 and r2 into temporary files separately;  (1)
   Sort the samples separately;                                         (2)
   Perform the intersection operations between
     the currently sorted samples, as well as between
     the currently and previously sorted samples;                       (3)
     { This is due to the full fulfillment implementation of
       the cluster sampling plan }
end.
```

Figure 4.4. Intersection Algorithm Used in Estimator Evaluation

Let n_{ji}, $n_{\bar{j}i} > 0$, be the number of tuples taken from r_j at stage i, and the current stage be the s_{th} stage. The cost of the first step is

$$c_1 \sum_{j=1}^{j=2} n_{js} + C_1 \qquad (4.2)$$

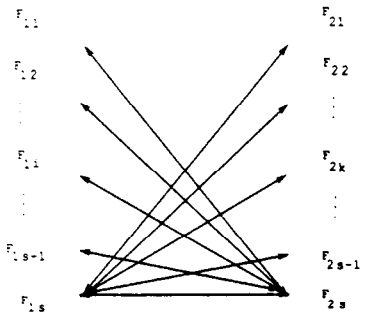
The second step is sorting, which is an external sorting, and its cost formula, after approximation, is

$$\sum_{j=1}^{j=2} (c_2 n_{js} \log n_{js} + c_3 n_{js}) + C_2 \qquad (4.3)$$

where log denotes the logarithm with base 2. Let F_{ji} denote the sorted file obtained from step (2) for relation r_j at stage i. j is either 1 or 2. If j=1 then $\bar{j}$=2 and vice versa. At stage s, not only an intersection operation has to be performed between F_{1s} and F_{2s} (i.e., between the current samples), but also between F_{1s} and all F_{2i}, $1 \le i < s$, and between F_{2s} and all F_{1i}, $1 \le i < s$. This is because of the full fulfillment implementation of the cluster sampling plan. Figure 4.5. shows the intersection operations to be taken at stages s. The cost for an intersection operation between F_{1i} and F_{2k} is $c_4 (n_{1i} + n_{2k}) + C_3 p_{1i,2k} + C_4$, where the first term is the time for reading and comparing tuples and the second term is the time for writing the output pages. $p_{1i,2k}$ denotes the number of pages written to the disk after an intersection operation between F_{1i} and F_{2k}. The total cost of step (3) at stage s is

$$\left\{ \sum_{j=1}^{j=2} \sum_{i=1}^{i=s-1} \left[c_4 (n_{ji} + n_{\bar{j}s}) + C_3 p_{ji,\bar{j}s} + C_4 \right] \right\} + c_4 (n_{1s} + n_{2s}) + C_3 p_{1s,2s} + C_4$$

$$= c_4 [N_{1s-1} + N_{2s-1} + s (n_{1s} + n_{2s})] + C_3 p_s + (2s-1) C_4 \qquad (4.4)$$



F_{1i} <-> F_{2k} : an intersection operation is to be performed between two sorted files F_{1i} and F_{2k}.

Figure 4.5. Intersection Operations to Be Performed at Stage s

where N_{ji} denotes the total number of tuples that are sampled from relation r_j from stage 1 through stage i, i.e., $N_{ji} = \sum_{k=1}^{k=i} n_{jk}$ and p_s denotes the total number of output pages to the disk at stage s. Therefore, the total cost of an intersection estimator evaluation at stage s is

$$IntersectCost\ (r_0, r_1, r_2, f, sel^+)$$
$$= \left\{c_1 \sum_{j=1}^{j=2} n_{js} + C_1\right\} + \sum_{j=1}^{j=2} (c_2 n_{js} \log n_{js} + C_3 n_{js}) + C_2$$
$$+ c_4 [N_{1s-1} + N_{2s-1} + s\ (n_{1s} + n_{2s})] + C_3\, p_s + (2s-1)C_4 \qquad (4.5)$$

4.4 Join Operation

Basically, the join operation and its time cost formula are similar to the intersection operation and the intersection time cost formula since, as observed from Figure 4.6, each step is of the same form (i.e., equations (4.2), (4.3) and (4.4)). Clearly, the values of coefficients and constants will be different.

Algorithm Join (r_1, r_2)
begin
Write the sample tuples from r_1 and r_2 to temporary files separately; (1)
Sort the temporary files separately; (2)
Perform the join operation between the current samples and as well as between the current and previous samples; (3)
end.

Figure 4.6. Join Algorithm Used in Estimator Evaluation

4.5 Project Operation

The cost of the first step is equation (4.2), the cost of the second step is equation (4.3), and the cost of the third step is $c_1 n + C_1 p$, where c_1 is the cost of reading a tuple from the temporary file plus the cost of checking whether the tuple is a duplicate or not, and C_1 is the cost of writing a page to the disk.

Algorithm Project (r_1)
begin
Write the projected attributes of sample tuples to a temporary file; (1)
Sort the temporary file; (2)
Scan the temporary file and write distinct tuples with their occupancy into the output relation. (3)
end.

Figure 4.7. Project Algorithm Used in Estimator Evaluation

5. Preliminary Experimental Results

This section reports the preliminary experimental results about the performance of the implemented prototype DBMS. It is important to note that, here, we only report the performance of the time-control algorithm in terms of the risk taken, the number of stages it went through, the number of disk blocks sampled in the given time quota, and the percentage of the time quota that has actually been used in evaluating the estimator. We do not report the performance of the estimation, which is already reported in [HoOT 88] in preliminary form, and in [HouO 88] in its full extensive form.

The stage-by-stage query evaluation technique based on the full fulfillment cluster sampling plan has been incorporated into a prototype database management system, called ERAM [DFHO 86], which uses relational algebra expressions as its query language. The system is built on top of Unix 4.3 BSD operating system running on SUN workstations. We present experimental results using only a single relational algebra operator. Results of projection operation are not discussed in this section. Extensive experiments are still being performed. All experiments have been carried out in a stand-alone SUN 3/60 node of the network in a single user mode (to avoid the intervention of other processes). In what follows, each artificial relation instance has 10,000 tuples, with the tuple size of 200 bytes. That is, each relation instance consists of 2,000 disk blocks (1K bytes in each disk block) with 5 tuples in each disk block. Each disk block is a sampling unit from a relation. Tuples in a relation are randomly distributed.

In order to provide additional information about the time control strategy, the ERAM does not abort a query (stage), as it should do so in a hard time constrained environment, when the query overspends. The "ovsp" (representing the word "overspend") columns give the average amount of time overspent (in seconds) in those experiments where overspending has occurred. For a given query, *the overspent time* is the time needed to complete the very last stage that was aborted due to the expiration of the time quota. The *wasted* time is the time spent for the very last aborted stage or the time left which is too small to start another stage. This information may be important for those environments to which soft time constraints apply.

In each of the following tables, the columns "stages" denotes the number of stages that the query has gone through. The column "risk" denotes the risk of overspending (in percentage) at the second stage. The "ovsp" denotes the amount of time overspent in those experiments where overspending has occurred. The "utilization" denotes the percentage of the time quota that has been used "successfully" in evaluating the query, and the rest of the time quota is "wasted". The "blocks" denotes the number of disk blocks evaluated within the given amount of time quota (i.e., the overall sample size). Each row in the following tables is obtained using a different d_β value. For example, in the first row (with $d_\beta = 0$) of the selection table, (stages = 1.56, risk = 56, ovsp = 0.11, utilization = 63, blocks = 54) represents the fact that we have chosen to use $d_\beta = 0$, and the query, on the average, has gone through 1.56 stages; the corresponding risk was 56%; the number of disk blocks evaluated was 54, utilization rate (including overhead) of the time quota was 63%, and the amount of time overspent was 0.11 seconds for those tries in which overspendings has occurred. Every entry in any table has been obtained from 200 independent experiments on RA operators.

The coefficients (i.e., c_i) in the time-cost formula were assigned initial values based on (i) the experiments with the largest possible tuples (i.e., 1K bytes) in our DBMS, (ii) the selection formula containing two integer comparisons, and (iii) two join attributes for the join operator. The values of the coefficients were then adjusted during runtime.

A. Selection Operation

As observed from the table, when the d_β value gets larger (i.e., a larger sel_i^{i+} is assumed), the number of stages becomes larger, but the risk of overspending becomes smaller (and hence the utilization rate becomes higher). When the d_β value is 0, the risk

should be close to 50%, since $sel_i^{i+} = sel_1^{i-1}$ from equation (4.3). When the d_β value is high (e.g., $d_\beta = 72$), the increase in the overhead offsets the gain in the utilization rate, and thus, the overall sample size (i.e., "blocks" columns in the table) decreases. For a given risk β, the d_β value in the experiments is very high, compared to the corresponding value in the normal distribution table. The reason is that we have chosen to use the cluster sampling plan in our implementation for efficiency reasons, instead of the simple random sampling of points in the point space. The amount of time overspent decreases as the d_β value increases. The amount of time overspent is always small; this implies that the run-time estimation approach for the selectivity estimation and the adaptive time cost formulas for time cost estimation have worked reasonably well. The selection operation was run with a selection formula containing only one integer comparison and an assumed maximum selectivity of 1 at the first stage.

Time Quota = 2.5 seconds

	1,000 output tuples				
d_β	stages	risk	ovsp	utilization	blocks
0	1.56	56	0.11	63	54
3	1.73	43	0.09	71	61
12	2.62	26	0.05	92	81
24	3.56	4	0.03	98	84
72	4.12	2	0.02	98	83

	5,000 output tuples				
d_β	stages	risk	ovsp	utilization	blocks
0	1.64	52	0.09	70	48
3	2.08	27	0.07	83	57
12	3.01	4.9	0.06	98	67
24	3.10	4.1	0.03	98	68
72	3.11	0.3	0.02	98	68

Figure 5.1. Performance of the Time-Control Algorithm for the Select Operation

B. Intersection Operation

We have used $1/maximum(r_1, r_2)$ as the initial value of the selectivity of the operation. When the d_β value is 72, we have found that the amount of time left (i.e., ≈7%) was not enough for a further stage in the implemented full fulfillment cluster sampling plan, and thus the query evaluation was terminated. Similar phenomenon was observed in the Join operation when the d_β values were 24, 48, and 72. (This implies that the partial fulfillment sampling plan in [HoOT 88a] may have its place here to use the small amount of time left.) Even though the utilization rates for d_β values of 48 and 72 are (almost) the same, the number of sampled disk blocks becomes smaller, from 54.1 to 51.9, when d_β values change from 48 to 72; again, this is due to the increase in the overhead and the increase in the time complexity of Intersection (and Join) operation (please see [HoOT 88a]).

C. Join Operation

In this experiment, we have assumed a selectivity of 0.1 at the beginning (compared to the actual selectivity $70{,}000 / 10{,}000^2 = 0.0007$) since, if the maximum selectivity of 1 were assumed, the sample size was so small (e.g., 4 disk blocks) at the first stage that the system clock did not provide enough accuracy in measuring the time spent. The join operation used in the experiment has only one join attribute.

Time Quota = 10 seconds

	10,000 output tuples				
d_β	stages	risk	ovsp	utilization	blocks
0	1.56	44	0.18	73	41.8
12	1.74	26	0.17	82	47.9
24	1.85	15	0.12	88	51.2
48	1.97	3.0	0.11	94	54.1
72	2.00	0	0.00	93	51.9

Figure 5.2 Performance of the Time-Control Algorithm for Intersection Operation

	70,000 output tuples				
d_β	stages	risk	ovsp	utilization	blocks
0	1.59	41	0.19	71	25.9
12	1.94	5.3	0.18	91	28.4
24	2.00	0	0.00	90	27.5
48	2.00	0	0.00	83	24.7
72	2.00	0	0.00	77	22.1

Figure 5.3 Performance of the Time-Control Algorithm for Join Operation

References

[AbGM 88] Abbott, R., and Garcia-Molina, H., "Scheduling Real-time Transactions", Proc., VLDB 1988 (Also in ACM SIGMOD RECORD, 1988).

[Coch 77] Cochran, W.G., "Sampling Techniques", Third Ed. John Wiley & Sons, 1977.

[Chri 83] Christodoulakis, S., "Estimating Record Selectivities", Information Systems, Vol 8, 1983.

[DFHO 86] Datta, A., Fournier, B., Hou, W-C., and Ozsoyoglu, G., "The Implementation of SSDB", Proc. 3rd SSDB workshop, Luxembourg, July 1986.

[Good 49] Goodman, L. A., "On the Estimation of the Number of Classes in a Population", Ann. Math. Stat., Vol. 20, 1949.

[HoOT 88] Hou, W-C., Ozsoyoglu, G., Taneja, B., "Statistical Estimators for Relational Algebra Expressions", ACM PODS Conference, March 1988.

[HoOT 88a] Hou, W-C., Ozsoyoglu, G., Taneja, B., "Processing Aggregate Relational Queries with Hard Time Constraints", CWRU Tech. Rep. CES-88-26, Dec. 1988.

[HouO 88] Hou, W-C. Ozsoyoglu, G., "Statistical Estimators for Aggregate Relational Algebra Expressions", May 1988.

[JoWi 82] Johnson, R., and Wichern, D., "Applied Multivariate Statistical Analysis", Prentice Hall 1982.

[Liu 68] Liu, C.L., "Introduction to Combinatorial Mathematics", McGraw-Hill, 1968.

[MuDe 88] Muralikrishna, M. and Dewitt, D., "Equidepth histograms for estimating selectivity factors for multi-dimensional queries", ACM SIGMOD 1988.

[PSCo 84] Piatetsky-Shapiro, G. and Connell, G., "Accurate Estimation of the number of tuples satisfying a condition", ACM SIGMOD conference, 1984.

[OzHO 88] Ozsoyoglu, G., Hou, W-C., Ola, A., "On Database Systems for Programming Logic Controllers", 1988.

[Rowe 85] Rowe, N. C., "Antisampling for Estimation : An overview", IEEE TSE Oct. 1985.

[SACL 79] Sellinger,P., Astrahan, M., Chamberlein, D., Lorie,R., Price, T., "Access path selection in a relational database management system", ACM SIGMOD 1979.