A Black-Box Approach to Query Cardinality Estimation

Tanu Malik
Dept. of Computer Science
Johns Hopkins University
3400 N. Charles St.
Baltimore, MD 21218
tmalik@cs.jhu.edu

Randal Burns
Dept. of Computer Science
Johns Hopkins University
3400 N. Charles St.
Baltimore, MD 21218
randal@cs.jhu.edu

Nitesh Chawla
Dept. of Computer Science
and Engineering
University of Notre Dame
Notre Dame, IN 46656
nchawla@cse.nd.edu

ABSTRACT

We present a "black-box" approach to estimating query cardinality that has no knowledge of query execution plans and data distribution, yet provides accurate estimates. It does so by grouping queries into syntactic families and learning the cardinality distribution of that group directly from points in a high-dimensional input space constructed from the query's attributes, operators, function arguments, aggregates, and constants. We envision an increasing need for such an approach in applications in which query cardinality is required for resource optimization and decision-making at locations that are remote from the data sources. Our primary case study is the Open SkyQuery federation of Astronomy archives, which uses a scheduling and caching mechanism at the mediator for execution of federated queries at remote sources. Experiments using real workloads show that the black-box approach produces accurate estimates and is frugal in its use of space and in computation resources. Also, the black-box approach provides dramatic improvements in the performance of caching in Open SkyQuery.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*distributed databases, query processing*; H.2.8 [**Database Management**]: Database Applications—*scientific databases*

General Terms

Design, Performance

1. INTRODUCTION

Database optimizers employ a bottom-up approach to query optimization. They require cardinality estimates[1] in order to obtain cost estimates for various query execution plans. Because execution plans are hierarchical, optimizers employ a constructive or bottom-up approach to obtain cardinality estimates at every level of the plan. In the bottom-up approach, the optimizer computes cardinality estimates of individual predicates and propagates these estimates progressively up the query plan, constructing cardinality estimates of operators, sub-queries, and finally the entire query.

Recently, several applications have emerged that interact with databases over the network and benefit from a priori knowledge of query cardinalities. These applications, which are the focus of this paper, do not need require query execution plans or estimates at every internal level. Examples include grid systems [26], proxy caching [3], replica maintenance [22, 23], and query schedulers in federated systems [2]. (Section 2.1 has more details). Accurate estimates of query cardinality improves performance through improved query efficiency, increased data availability, or reduced network traffic. Additionally, these applications are severely constrained in the amount of resources they can expend on query estimation. Resource constraints include storage space, processing time, and even the number of network interactions with the remote data sources.

The bottom-up approach of query optimizers for cardinality estimation is overkill for networked-database applications and bound to be prohibitively expensive. In addition, accuracy in query optimizers suffers from several modeling assumptions used in the bottom-up approach, such as conditional independence, up-to-date statistics, and uniform distribution of values in joins. Recent research initiatives [7, 30] relax these assumptions in query optimizers by including a self-tuning, self-correcting loop that refines estimates at each level using feedback from actual query and sub-query cardinalities. However, these approaches are either limited to range queries, because they are based on histograms, or require an intimate interaction with the database, which is unavailable to networked database applications.

We demonstrate that a simple "black-box" approach avoids the extra overhead of estimating cardinality for every sub-query and also avoids the drawbacks of the modeling assumptions used in the bottom-up approach. This black-box approach groups queries into syntactic families, called *templates*, and uses machine-learning techniques to learn the distribution of query result sizes for each family. Cardinality distributions are learned directly from points in a high-dimensional input space constructed from query attributes, constants, operators, aggregates, and arguments to user-defined functions. Thus, they are not subject to the inaccuracies that arise from modeling assumptions used in assembling an output estimate from sub-queries. We estimate query cardinalities based on the learned distribution and refine the distribution when the actual cardinality becomes available. This ongoing learning process creates a natural, self-correcting loop. This is a black-box approach in that it

[1]The term cardinality of a query refers to the number of rows in the query result whereas selectivity refers to the probability of a row being selected.

3^{rd} *Biennial Conference on Innovative Data Systems Research (CIDR)* January 7-10, 2007, Asilomar, California, USA.

estimates the cardinality of entire queries based only on the inputs and outputs of query processing.

Our treatment includes a comparative evaluation of several techniques that learn cardinality distributions in the high-dimensional input space. These include model trees, locally-weighted regression, and classification and regression. We previously reported on the use and efficacy of classification and regression for caching [17]. Our results show that different techniques present a time/space versus accuracy trade off.

A wide variety of distributed applications do not require fine-grained, sub-plan estimates and benefit from the increased accuracy and low space overhead of the black-box approach. We have developed these general techniques and deployed them in one such distributed application – The Open SkyQuery federation of Astronomy databases [28].

The Open SkyQuery federation presents a design space for which the black-box approach is uniquely suited. Open SkyQuery is a wrapper-mediator [27] system that allows scientists to cross-query heterogeneous, globally-distributed Astronomy databases. The mediator includes a scheduler and a proxy cache that caches portions of the federated data. On receiving a cross-query, the mediator partitions the query into components, one for each remote database. It then uses cardinality estimates of the component queries to schedule them across remote sites [19] and to make cache revocation decisions within an economic caching framework [18]. In both cases, cardinality estimates for the entire component query to a member database suffice. (At the member database, the local optimizer constructs a query execution plan using its own private data structures.)

The federated architecture mandates data-independent estimation; the system must operate without access to the underlying data, making estimates based on the observed workload – queries and their results – alone. Methods that rely on sampling or generating statistics, *e.g.* data-dependent histograms, are not tenable, because they require samples from Terabyte-sized databases in the federation. Scanning the database to build and maintain such statistics has been shown to be extremely costly [1].

The complex nature of Astronomy queries adds to the challenge of using only query workloads for cardinality estimation. Astronomy queries contain real-values attributes, multi-dimensional range clauses, and user-defined functions in select and join clauses. Furthermore, attributes are highly correlated with each other (invalidating conditional independence assumptions). Known techniques estimate cardinality for queries that possess some of these characteristics, but not all.

The black-box approach differs fundamentally from bottom-up estimation. It corresponds to the *declarative* query specification just as the constructive approach corresponds to the *imperative* query execution plan. Thus, the black-box approach is not naturally suited for use within a database optimizer; it neither estimates the cardinality of sub-queries nor identifies opportunities for parallel execution and the ordering of operators within a query.

An experimental evaluation of black-box cardinality estimation shows the suitability of the technique for distributed applications. We evaluate it using multiple learning algorithms against an Open SkyQuery workloads of 1.4 million queries. Results indicate that black-box estimates require data structures of tens or hundreds of kilobytes, produce estimates quickly, and that the accuracy of these estimates greatly improves the performance of caching in Open SkyQuery.

2. RELATED WORK

To motivate the black-box approach, we present a series of networked applications that require accurate cardinality estimates from remote data sources. Then, we review prior research on data-independent cardinality estimation, examining its suitability to such applications.

2.1 Applications

Recently, several caching and grid-based applications have emerged that require cardinality estimates of queries. Often, these assume the presence of knowledge systems to provide estimates. In approximate data caching [22], sources cache exact values and caches store approximate values near the client. In presence of updates at the server, approximations of cached values become invalid. On invalidation, new approximations (based on the degree of precision desired) are either propagated by sources to the caches or alternatively demanded by queries. The goal is to minimize the overall network traffic by lowering the cost of push by the server and cost of pull by the queries. For non-aggregate queries, the exact computation of the pull cost of a query requires the knowledge of selectivity of a query. This is also true in other push-pull models of data dissemination [5].

In adaptive caching environments, cache state is adjusted dynamically as workload changes. Cache replacement algorithms often compute the benefit of caching a data object in which the benefit is based on various statistics, one of them being the size of query result against that object. Systems such as DBProxy and Bypass-Yield Caching [18] assume that query result sizes are known a-priori. The efficacy of these techniques depends on the correctness of this assumption.

In GridDB [16], a data-centric view of the grid has been proposed in contrast to a process centric view of Condor [15] and Globus [10]. Due to the long nature of scientific queries, Interactive query processing (IQP) is considered an essential feature of GridDB [16]. In IQP, users often want to know just the size and cost of running expensive jobs and base further jobs on these answers. This requires accurate cardinality estimates.

In our experience with the Sloan Digital Sky Survey, we have witnessed several load balancing and scheduling applications [11] in which such knowledge is required. In this paper, we have considered one such scientific application, the Open SkyQuery federation of Astronomy databases, which needs cardinality estimates to make scheduling decisions as well as to populate its proxy cache. For the caching system, we recently reported a 50% degradation in absence of accurate estimates [17].

2.2 Data-Independent Cardinality Estimation

We review the prominent data-independent methods for cardinality estimation and learning in optimizers and consider their applicability to the above applications. These methods are derived from the concept of self-tuning in which queries are estimated using learned distributions and the actual result sizes of queries provide feedback to refine these distributions. Current work on self-tuning is limited

in that either: (1) it restricts the classes of queries that may be estimated; or, (2) techniques are closely tied to the optimizer. To the best of our knowledge, ours is the first technique to support general queries in a data-independent fashion.

Most self-tuning research has been conducted in the context of histograms, which limits the techniques to range queries. Histograms cannot be used for point queries and user-defined functions.

Aboulnaga and Chaudhuri [1] use the query feedback approach to build self-tuning histograms. The attribute ranges in predicate clauses of a query and the corresponding result sizes are used to select buckets and refine frequency of histograms initialized using uniformity assumption. Both single and multi-dimensional (m-d) histograms are constructed by this method. Higher accuracy on m-d histograms is obtained by initializing them from accurate single dimensional histograms.

STHoles [8] presents the technique most similar in spirit to our black-box approach. STHoles refines the layout and frequency of existing histogram buckets by allowing nesting of buckets. As queries to a region increase, new buckets are initialized within existing buckets to improve the accuracy. The algorithms use very detailed query feedback from the query execution engine, examining the distribution of data within query results. STHoles works well in refining existing histograms and also in building histograms in a data-independent fashion, based on queries and their results alone.

ISOMER [29] constructs histograms that are correct and consistent with query feedback. It utilizes the maximum entropy principle to select a distribution that has the maximum information among a set of distributions each of that is consistent with the query feedback.

CXHist [14] builds workload-aware histograms for selectivity estimation on a broad class of XML string based queries. XML queries are summarized into feature distributions and their selectivity is quantized into buckets. Finally, it employs a naive-Bayes classifiers to compute the bucket to which a query belongs. The naive-Bayes approach assumes conditional independence among the features within a bucket.

User-defined functions present a different challenge, because the function obfuscates the relationship between the underlying data distribution and the query result size. UDFs demand an approach that learns the output result size distribution directly. He *et al.* [13] define a self-tuning framework for defining the cost model of user-defined functions (UDF). To estimate the cost of a query, they examine the cost of the k-nearest neighbors to that query in the multi-dimensional space defined by the function arguments. The estimated cost is computed as a weighted average of the cost of these neighbors. They do not specifically estimate result sizes (it is left as future work), but their technique is suitable. Our black-box approach extends these techniques, learning in a richer space that includes attributes, constants, aggregates, and operators. We also take a different approach to learning, using either regression trees or classification and regression, that constructs a model on compact, summary data structures.

DB2's learning optimizer (LEO) [30] provides the most widely-applicable learning technique, which includes learning for join predicates, keys created by the DISTINCT and GROUP BY clause, derived tables, user-defined functions, etc. However, the cardinality estimates created by LEO are obtained by correcting modeling errors at every level of the query execution plan. Thus, LEO is tightly coupled to a DB optimizer. The corrections provided by LEO do not adapt to or learn data distributions or result size distributions that vary with input parameters.

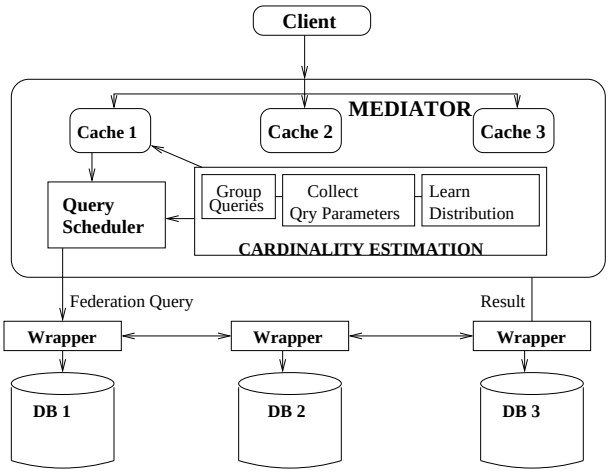


Figure 1: Abstract Architecture of Open SkyQuery

3. CASE STUDY: OPEN SKYQUERY

In this section, we describe the Open SkyQuery federation of Astronomy archives. In particular, we motivate the need for accurate cardinality estimates and define the resource constraints placed on the estimator.

Figure 1 shows the abstract architecture of the Open SkyQuery federation. Open SkyQuery uses a wrapper-mediator architecture [27]. Users submit federated queries (that access multiple databases), which the mediator divides into component sub-queries that are executed at their respective sites. The mediator includes a scheduler, which decides the order in which various member sites are to be visited for sub-query execution. This order is decided so as to minimize network traffic and query response time. Because scientific queries are long-running and data-intensive, the primary measure in the scheduling decision is the cardinality of each sub-query. In earlier versions of the system, the scheduler polled each database for the actual cardinality of its component query. This is a bottleneck as it increases average query response time.

The mediator includes an adaptive cache that replicates objects, such as columns (attributes), tables, or views, from member databases so that sub-queries to the member database may be served locally by the mediator. This reduces the network bandwidth usage of data-intensive queries. The cache uses the cardinality of each sub-query to decide when to load and evict database objects. More specifically, cardinality specifies the network cost of the current query in a rent-to-buy economic framework [18].

The scheduler needs cardinality estimates for each sub-query. The cache may also need estimates for a combination of sub-queries when distributed joins are specified in the query. In either case, the mediator has to estimate query cardinalities using a small amount of space. Each mediator in the system must generate estimates for all the federation's member databases, using only local storage.

In addition, estimation models are to be learned with minimum

```
select [select−list]
from PhotoPrimary as P
join fGetNearbyObjEq(185,−0.5, 1) as D
on P.objid = D.objid
where (P.r between 8.0 and 11.0)
order by distance
```

```
select [select−list]
from Star where
ra between 119.72417 and 122.72417 and
dec between 67.64444 and 70.64444 and
r between 12.0 and 17.0 and
((g−r) >= 0.37) and ((g−r) <= 0.54) and
((r−i) >= 0.07) and ((r−i) <= 0.15)
```

```
select [select−list]
from Galalxy as G
join fGetNearbyObjEq(334.36013, 0.266444, 17) as N
on G.objid = N.objid
where
((G.flags_r & 0x10000000) != 0) and
((G.flags_r & 0x8100000c00a4) = 0) and
(((G.flags_r & 0x40000) = 0) or (G.psfmagerr_r <= 0.2)) a
(((G.flags_r & 0x10000) = 0) or (G.flags_r & 0x1000) = 0)
```

Figure 2: Complex user queries on member databases in the Open SkyQuery federation

access to data. In general, the mediator is far from the member databases. This creates an access barrier between the mediator and the servers. Management boundaries create privacy concerns making it difficult to access the databases to collect statistics on data. The data is mostly accessed via the restricted Web-services wrapper interface (Figure 1). Even if the mediator can store summary statistics, the member sites may choose not to invest their resources in the I/O-intensive process of collecting statistics on all of the data [9].

The continuous stream of user queries and their results at the mediator provide indirect access to queries and data and can be used for learning output cardinality models. Queries in the Open SkyQuery federation are complex, which makes learning of estimation models a non-trivial task. A typical query is a conjunction of multiple range and user-defined function clauses in the predicate expression, as well as user-defined functions in the join clause. We consider three real queries taken from a member database of the Open SkyQuery federation. These queries exemplify the complexity of learning an estimation model from queries (Figure 2). (The select list is suppressed as it does not influence cardinality.) The first query shows the combined use of range clauses and user-defined functions, the latter occurring in both the join and the predicate clause. The user-defined function *fgetNearByObjEq* returns a temporary table of nearby objects. Its arguments vary depending upon whether a circular, a rectangular, or a polygonal region is selected. The second query is a five-dimensional range query in which the first three range clauses are on database attributes and the last two are on temporary attributes created from subtracting values from two database attributes. The third query is function join with bit operators in the predicate clause. Histograms, such as STGrid [1] and STHoles [8], efficiently learn estimation models when query workloads consist predominantly of range clauses on database attributes. Because they build multi-dimensional histograms, they can also estimate range clauses with mathematical operators on a combination of attributes (such as g - r $>=$ 0.37 in the second query), but cannot update the histogram bucket boundaries from this combined information. Estimation models based on nearest neighbor methods have been shown to be more effective for user defined functions [13]. However, Open SkyQuery workloads consist of a combination of range (simple or complex) and user-defined function clauses.

```
Select # [select−list]
From R,S,T
Where R.r ? S.s
and   R.r ? T.t
and   S.b ? % and ? %
and   R.a * R.a ? %
and   T.c & Ox0011 ? %

     # = {TOP,NOTOP,MAX}
     ? = {<,>,=,!=}
     % = {ints,reals}

a) Template and its parameters
```

```
    Parameter Space           Cdn
<----------------------->   <------->
{#,?,?,?,%,?,%.?,%,?,%       Cdn_Value}

{TOP,=,=,>,50,<,100,<,0.1,=,0      10}
{NOTOP,=,=,>,23,<,45,>,0.6,!=,1   250}
{NOTOP,!=,=,>,50,<,80,<,0.9,=,0  2000}
{TOP,=,=,>,20,<,168,>,0.8,=,1     790}
{NOTOP,!=,=,>,59,<,63,>,0.3,!=,0 3890}

(b) Cardinality (Cdn) distribution
     of the parameter vectors
```

Figure 3: Learning in templates

We conclude that the federation needs a light-weight, data-independent, and general estimation mechanism, which accurately predicts query cardinality.

4. THE BLACK-BOX APPROACH

Cardinality estimation treats query evaluation, and optimization as a black box, examining the input query and the cardinality of its results alone. Data distributions are unknown, owing to data-independence requirements, and cardinality estimates at every sub-query level are not required by our applications. The approach first groups queries into syntactic families. It then learns cardinality distributions directly as a function of query parameters, such as attributes, operators, constants, aggregates, and user-defined functions and their arguments. We apply several machine learning techniques, including classification and regression, model trees, and locally-weighted regression. These result in concise and accurate models of the cardinality distribution. When a query arrives, cardinality is estimated using the model. When the query is executed, the query parameters and its result together update the model. This results in feedback-based learning.

The black-box approach works best when workload satisfies certain criteria. We discuss the most important criteria that enable learning from workload, improve accuracy, and limit space overhead in the black-box approach. An example at the end of the section illustrates the learning process for complex queries.

4.1 Estimating Query Cardinality

Grouping Queries: The black-box system groups queries into *templates*. A template τ over SPJ (Select-Project-Join) queries and user-defined functions (UDFs) is defined by: (a) the set of objects in the `from` clause of the query (objects imply tables, views, or tabular UDFs) and (b) the set of attributes and UDFs occurring in the predicate expressions of the `where` clause. Intuitively, a template is like a function prototype. Queries belonging to the same template differ only in their parameters. For a given template τ, the input parameters (Figure 3(a)) are: (a) constants in the predicate expressions, (b) operators used in the join criteria or the predicates, (c) arguments in table valued UDFs or functions in predicate clauses,

and (d) bits which specify if the query uses an aggregate function in the select clause. A template's parameters do not include attributes in the select clause; the parameters of a template are only those features in a query that influence/determine its cardinality.

Collecting Query Parameters: The parameters of a query that influence its cardinality form a vector. Figure 3(b) shows vectors with parameters and cardinalities from a few queries. For a template, the parameter space refers to all the possible combinations of parameters, each chosen from their respective domain. A query belonging to a template refers to one of these combinations and has a corresponding yield in the range $\mathcal{R} = (0, max(t_i))$ in which $max(t_i)$ is the maximum yield of a query in a given template, *i*.e., the size of a relation for a query to a single relation and the size of the cross product for a join query.

Once the template is created, only the template parameter values and the corresponding cardinalities are retained. The parameter values represent a point in the input space associated with the cardinality observed at that point; *i.e.,* the cardinality is a function of the multi-dimensional point in the parameter space represented by the parameter values. The system has no knowledge of the exact role of the parameters in query execution. In a template, the parameter values represent a cardinality distribution that the system uses to estimate the cardinality of future queries to the same template. Precisely, the actual cardinality distribution of a template τ over n queries is the set of pairs

$$\tau_D = (p_1, y_1), (p_2, y_2), \ldots, (p_n, y_n) \quad (1)$$

in which p_i is the parameter vector of query q_i, and y_i is its cardinality. Because of the high dimensionality of the parameter space, the system employs machine learning techniques to approximate the cardinality distribution.

Approximating the distribution: Most attributes in the Open SkyQuery databases are numeric and record various physical properties of the astronomical bodies. Thus, query parameters are also numeric, making regression the natural technique for cardinality estimation. Regression models cardinality as a function of the values of a multi-dimensional parameter vector [6]. The simplest form of regression is linear regression [20] in which the cardinality y_i is modeled as a linear function of the multidimensional vector $p_i = p_{i,1}, \ldots, p_{i,n}$ as

$$y_i = w_o + w_1 p_{i,1} + \ldots + w_n p_{i,n} \quad (2)$$

in which the coefficients $w_o, w_1, \ldots, w_n$ are estimated using the least squares criterion. However, a naive application of regression introduces a high learning bias, because a linear model cannot capture the non-linearity of the parameter distribution. We apply three learning techniques, namely classification and regression, model trees, and locally-weighted regression. All of these techniques partition the input space and use regression within the partitions in order to estimate cardinality. They differ in the criteria and methods they use for partitioning. They also differ in their computational complexity, space requirements, generality, and ease of use. Specifically, the techniques show a trade-off between size and accuracy. Based on the requirements of a distributed application, one of these techniques may be chosen. In the Open SkyQuery, we have opted for classification and regression because of its low computational complexity and space requirements.

1. *Classification and Regression*: In classification and regression, the system groups queries within a template into classes and transforms the distribution in equation 1 to an approximate, class distribution

   $$\tau_C = (p_1, c_1), (p_2, c_2), \ldots, (p_n, c_n) \quad (3)$$

   in which $c_i = C_F(y_i)$ and C_F is a classification function that transforms the numerical cardinality into a nominal class value (Figure 3(c)).

   The system learns this class distribution using decision trees [25]. Decision trees recursively partition the parameter space into axis orthogonal partitions until there is exactly one nominal class (or majority of exactly one class) in each partition. The partitioning is based on information gain of parameters so as to minimize the depth of recursion, *i.e.,* the parameter attribute with the highest information gain is chosen as the partitioning attribute. The system uses decision trees as they are a natural mechanism for learning a class distribution in the parameter space in which independence among parameter values cannot be assumed.

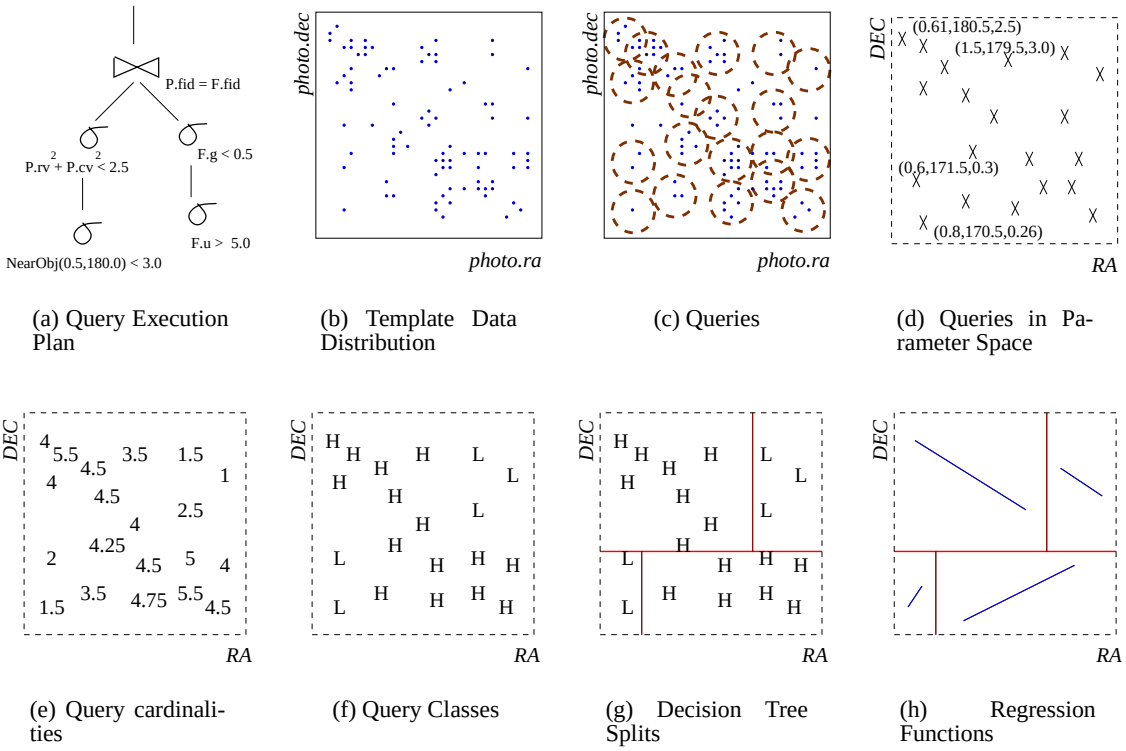
   By learning the τ_C *classes* of yields and not the τ_D *values* of cardinalities, there is some loss of information. The system regains some of this lost information by constructing a linear regression function within each class. A class specific regression function gives cardinality values for different queries that belong to the same class.

   Finally, the system uses k-means clustering as the classification function C_F in which k is the number of classes and is a dynamically, tunable parameter. Several techniques [12, 24] can be used as a wrapper over k-means to find a suitable k from the lowest and highest observed yield value, or it may be chosen based on domain knowledge.

2. *Model Trees*: Model trees store a piece-wise linear approximation of τ_D. Like decision trees, they use a divide-and-conquer principle to recursively partition the parameter space into axis orthogonal partitions until cardinality values can be accurately predicted using a linear regression model. The partitioning criteria choose an attribute that maximizes the expected error reduction, in which the standard deviation of the cardinality values serves as the error measure. We consider pruned model trees, as they provide higher accuracy over non-pruned trees, and are also much simpler. However, pruning reduces accuracy as they introduce discontinuities between adjacent linear models at the leaves of the pruned tree. A smoothing process [31] is applied to compensate for the these sharp discontinuities and regain some accuracy

3. *Locally-weighted Regression*: Given a test parameter vector, locally-weighted regression weighs all the training parameter vectors according to a distance metric to the test parameter vector. It then performs a linear regression on the weighted vectors. Training vectors close to the test vector receive a high weight; those far away receive a low one. In other words, the linear regression model is constructed on-the-fly for the particular test vector at hand and is used to predict the instance's class value. The distance function is chosen as the inverse of the Euclidean distance function. Since all training is done at prediction time, *i.e.,* the vectors are scanned to compute the weights, locally-weighted regression is much slower than the models described above.

Refinement: Once the system has used the estimate and served the query result, the size of the result is used as a feedback to refine



(a) Query Execution Plan

(b) Template Data Distribution

(c) Queries

(d) Queries in Parameter Space

(e) Query cardinalities

(f) Query Classes

(g) Decision Tree Splits

(h) Regression Functions

Figure 4: Applying the black-box to SDSS queries

the estimation models. Refinement includes changing the partitioning of the input space and updating the regression function in each partition. For classification and regression, repartitioning the input space means rebuilding the decision tree. Similarly, for model trees, the model tree must be rebuilt. Owing to their compact size, rebuilding the learning models is not expensive and could be done for every query. Our experiments show that it takes less than 20 seconds over thousands of queries(Section 5). However, we elect to batch updates, rebuilding trees after collecting a fixed number of queries in a template. We are currently exploring methods of recursive least-squares error and incremental induction trees to update the classification and regression model in situ. However, we hold that incremental update is not a critical feature of learning models for black-box cardinality estimation.

Locally-weighted regression is an incremental technique. Each new query becomes part of the training vectors. This incremental advantage comes at a cost. Regression functions are built on demand for each query, making locally-weighted regression more expensive at runtime. In a sense, incremental advantage is achieved by deferring the model update, processing updates at query time as opposed to when the feedback occurs.

4.2 Learning from Queries

Cardinality distributions that may be learned accurately from query workloads alone possess certain properties, such as syntactical locality and data locality. Additionally, learning techniques are robust to workload properties that are a challenge for traditional optimizers, such as multi-dimensionality, multi-way joins, and user-defined functions. We highlight the properties of the Open SkyQuery workload and how it interacts with learning techniques.

High Template Concentration: The black-box approach works best when a small number of templates characterize most queries. As it happens, most real-world application workloads consist of queries created through pre-designed interfaces such as forms and prepared statements [4]. Such an interface naturally results in queries within the same template that vary only in a parameters (in the predicate and join clauses), operators, and aggregates. This makes templates analogous to function prototypes.

The Open SkyQuery workload exhibits extremely high template concentration. In a typical month's workload, there are 1.4 million queries. The top 77 templates capture 97% of these queries. Open SkyQuery templates originate from static forms as well as repeatedly used user programs. These user programs are dynamic in nature – they are debugged and modified to capture new science questions – but templates change slowly over time. Nevertheless, the high concentration allows the black-box approach to provide high accuracy at low space overhead.

High Parameter Value Concentration: Queries of a given template have parameter values that effectively represent a very small part of the entire attribute domain. They often involve nearest neighbor or region searches, resulting in consecutive queries with small parameter differences. Thus, there is high concentration in parameter values for a large number of queries of a single template. This gives the system sufficient sample queries to learn a distribution.

High Dimensionality of Queries: Queries are said to be multi-dimensional if they possess more than one predicate clause in the

query, where predicate clauses can be range clauses or user-defined functions. Query dimensionality also increases if queries allow specification of user-defined functions in join clauses. High dimensionality forces the use of a multi-regression model [6]. This is the primary reason for our choice of regression as the basic learning technique for estimating query cardinalities.

Selection of Relevant Parameters: While decision tree learning algorithms are designed to learn which are the most appropriate parameters to use for making classification decisions, adding irrelevant or distracting parameters to the training set often "confuses" the learning algorithm. This is especially true of parameters in user-defined functions which do not contribute to query cardinality estimation. We currently use domain knowledge to prune such parameters. We are experimenting with instance-based learning methods for relevant parameter selection. Similarly, it is important that the relevant parameters encode the maximum information available, *i.e.,* include both syntactic and semantic information. For example, a range predicate clause of form "col BETWEEN constant1 and constant2" is transformed as (constant1,constant2-constant1). The difference captures the distance between the two constants and has more semantic information than other feature vectors, such as (constant1,constant2). This is also true for bit operators where an expression of the form "col bit_op const eq_op const", in which $bit\_op \in (|, \&)$ and $eq\_op \in (=, \neq)$, is transformed to an exact value of the column "col".

Learning from queries is not space efficient when the underlying distribution is uniform. In the classification and regression model, the decision tree recursively partitions for each training instance to take up space in proportion to the number of instances encountered. (Accuracy is not a concern, as regression learns the uniform distribution.) It is easy to detect such distributions when there are primary key attributes in the predicate expression and the system avoids learning such distributions.

4.3 An Illustrative Example

We illustrate the black-box approach using an example Astronomy template from the SDSS workload. This example template represents most typical templates in the SDSS workload. In particular, we show (a) how the query cardinality can be estimated without creating/learning statistics at every sub-query level, and (b) how the underlying distribution can be *indirectly* learned using only queries and their result. We compare it with the alternative bottom-up approach commonly employed by query optimizers, which uses histograms (single or multi-dimensional) to approximate base table distributions.

Consider the following template

```
SELECT * FROM
Photo P,
Field F
WHERE P.fid = F.fid and
P.NearObj(@ra,@dec,@radius) and
P.rv^2 + p.cv^2 <= @velocity and
F.u > @in1 and F.g < @in2
```

The relation `Photo` stores attributes about astronomical bodies. These attributes are its id (`objid`), spatial location (`ra` and `dec`), and velocities (`rowv`,`colv`). The relation `Field` stores the attributes which measure the radiation intensity (`a_u`, `a_g`) of a photometric field (`fid`). Given a field of certain radiation intensity

Number of queries	1,403,833
Number of query templates	77
Percent of queries in top 15 templates	87%
Network traffic from all queries	1706 GB
Percent traffic from top 35 templates	90.2%

Table 1: SDSS Astronomy Workload Properties

(`F.u` > `@in1` and `F.g` < `@in2`), queries to this template select those objects from photo that are spatially close (within `@radius`) of a point (`P.NearObj(@ra,@dec,@radius)`) and have a certain velocity ($\texttt{P.rv}^2 + \texttt{p.cv}^2 <= \texttt{@velocity}$).

To estimate cardinality of queries to the above template, the bottom-up approach considers the query execution plan (QEP) shown in Figure 4(a). Sub-estimates are needed at every level of the plan to correctly compute the cardinality of the query at the top level. The large number of these sub-estimates (in this case 5) increase optimization costs adding to the overhead of the bottom-up approach. Further, sub-estimates are calculated by assuming conditional independence between attributes. Incorrect propagation of these sub-estimates up the plan lead to inaccuracies.

Complex queries make it difficult to translate from an underlying data distribution to a cardinality estimate. User-defined functions present an extreme case, because they are totally opaque to a query optimizer. Figure 4(b) shows the underlying data distribution of attributes `@ra` and `@dec`. The query uses function `NearObj` against this distribution, which computes a spatial range query. Figure 4(c) annotates the data distribution with ranges evaluated by the function. Such a range query is estimable using multi-dimensional histograms. However, the function prevents cardinality estimates from being drawn from a histogram. It turns what is in fact a range query into a point query in the parameter space (Figure 4(d)) with cardinality that is a function of the input parameters.

We illustrate the black-box approach using classification and regression as our learning technique. We demonstrate how to estimate function `NearObj` from the point queries. For the sample query, the actual learning occurs in a twelve-dimensional input space over all operators, constants, and function parameters. For simplicity, we visualize just the function in its two spatial dimensions.

The high variance in cardinality for queries in this template gives the black-box approach a cardinality distribution to learn. The training/feedback data consists of the observed cardinality of completed queries. Figure 4(e) shows the logarithm of the cardinality at each point. It learns by observing that some queries have high (H) cardinality value (log values greater than 3.5) and some low (L) (Figure 4(f)). It classifies this distribution using decision trees, which split on specific values of the parameters, dividing into regions of purely high cardinality and purely low cardinality. We show splits into two cardinality classes (Figure 4(g)). In practice the technique uses between 4 and 8 yield classes, depending upon the natural clustering of cardinality values in the workload. Figure 4(g) shows an initial split on the dec parameter and then a split on the ra parameter. The decision tree itself can be used for estimation, but we improve upon it by using regression. In each decision tree leaf node, we look at the original cardinality values for queries and approximate this distribution through linear regression. We show this pictorially in Figure 4(h).

Template	Number of Queries	% Domain Requested	Dimensions in Feature Vector	Available Index	Query Semantics
T1	23343	5.1 x 10e-7	4	1	Range query on a single table
T2	103148	4.3 x 10e-27	6	0	Function query on a single table
T3	761605	2.9 x 10e-18	4	0	Function query on a single table
T4	4142	4.7 x 10e-4	3	0	Function query on 2 joined tables
T5	68603	3.5 x 10e-14	4	1	Equality query on a single table
T6	3176	1.3 x 10e-22	5	0	Function query on 3 joined tables

Table 2: Six important query templates in the SDSS workload

5. EXPERIMENTAL RESULTS

We first describe the workload and quantitatively evaluate it with respect to the properties (Section 4.2) that enable effective use of the black-box approach. We then measure the accuracy and efficiency of the black-box approach under various machine learning techniques by comparing it with a bottom-up approach. In particular, we show that, without constructing sub-estimates at every level, the black-box approach accurately predicts the cardinality of incoming queries and constructs learning models with minimum space and time overhead. We also show the impact that accurate cardinality estimates on the network performance of the caching module in Open SkyQuery. This demonstrates the efficacy of the black-box approach in distributed environments. All experiments were performed on a IBM workstation with 1.3GHz Pentium III processor and 512MB of memory, running Red Hat Linux 8.0.

5.1 Workload Properties

We took a month long trace of queries against the Sloan Digital Sky Survey (SDSS) database. The SDSS is a major site in the Open SkyQuery federation. Our trace has more than 1.4 million queries that generate nearly two Terabytes of network traffic. An analysis of the traces reveal that an astonishingly small number of query templates capture the workload (Table 1). A total of 77 different query templates occur in all 1.4 million queries. Further, the top 15 of these templates account for 87% of the queries, and the top 35 account for 90% of the total network traffic. Thus, the SDSS traces exhibit a remarkable amount of template concentration.

Individual templates exhibit the desirable workload properties described in Section (4.2). To quantify workload properties, we study the top six templates (in the number of queries) that account for 68.7% of the workload (Table 2). The **number of queries** shows the high concentration of queries in few templates. The **% domain requested** indicates the fraction of domain values accessed by queries as a percentage of all values in the database. The extremely low percentages indicate high parameter value concentration. The top templates also show that the high-dimensionality of queries contributes to learning. The **dimensions in feature vector** indicates the number of input parameters that are partitioned by classification and regression. This corresponds to the features of a template that provide information and indicate that many dimension must be used to accurately estimate this Astronomy workload.

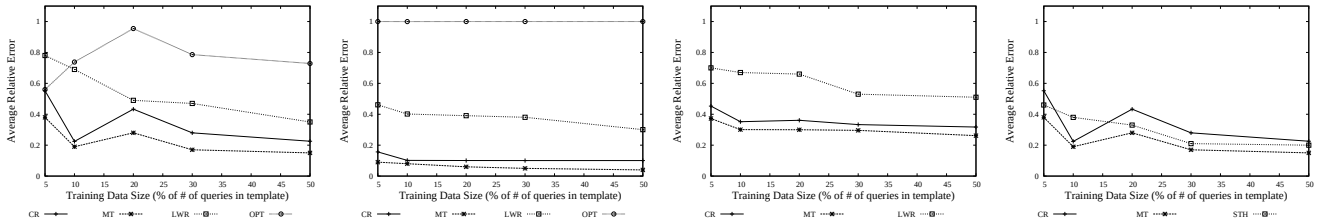
This is a representative one month workload from the SDSS database of the Open SkyQuery. We predict that workloads for other Astronomy databases show similar properties. Unfortunately, other large-scale workloads are not publicly available.

5.2 Metrics and Methodology

We show the accuracy and efficiency of the black-box approach over the entire workload of 1.4 million queries. We then zoom-in over the top six templates, which form a large percentage of the workload and show (a) the learning accuracy in each template and (b) the low space and time complexity of the relevant data structures in each template. For these templates, we also show the influence of various parameters in determining cardinality of queries.

Comparison Methods: We compare the accuracy of the black-box approach (**BB**) with, estimates from a commercial optimizer (MS SQL 2000) (**OPT**). Optimizer estimates in MS SQL 2000 are stored on a per attribute basis as a variation of max-diff histograms [21]. Commercial optimizers do not store estimates of sub-plans generated during optimizations, but use the base-table statistics and propagate them up using the assumption of conditional independence. While it is known that this introduces error in the optimizer, it was difficult to implement or obtain implementations of the latest research in the bottom-up approach, which relaxes these assumptions, because they are tightly integrated with source-code of proprietary optimizers [7, 30]. In the **OPT** method, histograms were constructed only for the attributes that were present in the query workload, using 5% sampling. The reader is directed to the documentation on MSSQL for creating and maintaining statistics [21]. For the black-box approach, we use the three models of regression: classification and regression (**CR**), model trees (**MT**), and locally-weighted regression (**LWR**) to obtain accuracy and efficiency measures. In addition, for simple templates, such template T1, with only range clauses in its predicate expressions, we implement self-tuning histograms [1] (**HIST**). **HIST** relaxes the assumption of conditional independence and constructs a histogram incrementally from queries and their results only. This technique only works for range clauses and not user-defined functions and, therefore, was not implemented on other templates.

Refinement and Parameters: To compare the overall accuracy and efficiency of the 1.4 million query workload, we use on-line refinement. In on-line refinement, a query is simultaneously issued to the **BB** and the **OPT** methods and the estimation error is recorded. In the **BB** method, if a template is found the query is estimated from the existing model, and finally the query and its result update the model. If a template is not found, the **OPT** estimate is taken as the initial estimate. Since there are few templates, this is done for few queries. The **OPT** estimate is taken as the default estimate that is always available in such distributed applications. To show the accuracy and efficiency over the top six templates, we switch to offline/batch refinement. In offline/batch refinement, we build an *a priori* model using training queries and then use the model to make predictions using testing queries. In addition, we keep the distribution of test queries the same as those of training queries. The size of the training set is varied from 5% to 50%.



(a) Template T1 (b) Template T3 (c) Template T4 (d) Comparison with self-tuning histograms

Figure 5: Accuracy results for the SDSS workload

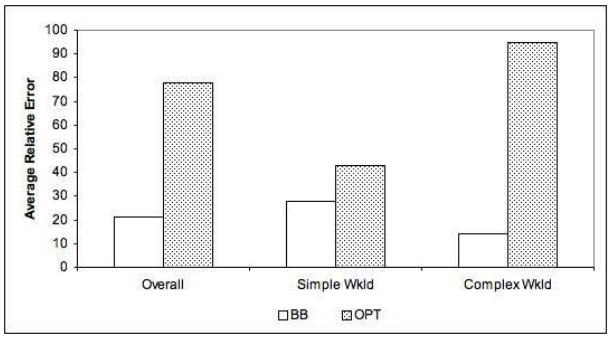


Figure 6: Overall Performance

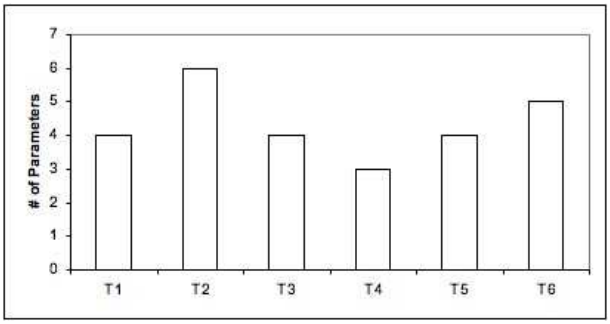


Figure 7: Influence of Parameters on Query Cardinality

Measuring Accuracy and Efficiency: To measure the overall accuracy of our approach, we use the average absolute relative error

$$\bar{E} = \frac{1}{N} \sum_{q \in W} \frac{\mid \rho(q) - \hat{\rho}(q) \mid}{\rho(q)} \tag{4}$$

in which $\rho(q)$ and $\hat{\rho}(q)$ are the actual and estimated yield of a query q in the workload W, and N is the total number of queries in W. We use the same error measure when measuring accuracy of all queries in a template in which case N corresponds to the total number of queries in a template.

The Open SkyQuery mediator requires different metrics to measure the effect that accuracy has on the performance of caching. For the caching module, we use the absolute error averaged over all queries as the error measure

$$\bar{E} = \frac{\sum_{q \in W} \mid \rho(q) - \hat{\rho}(q) \mid}{\sum_{q \in Q_i} \rho(q)} \tag{5}$$

in which $\rho(q)$ and $\hat{\rho}(q)$ are the actual and estimated yield of a query q in a given template. This absolute error corresponds well with the notion of total network traffic saved, the performance measure for the cache.

5.3 Accuracy

The black-box approach (with CR as the learning technique) is about 4 times more accurate than the optimizer when considering the entire workload (Figure 6). The overall improvement can be attributed to improved estimates for user-defined functions and complex expressions. The black-box approach learns from input parameters for which the optimizer is not able to create distributions from base statistics. We further divide the workload into simple and complex templates. Simple templates consist primarily of queries with range and join clauses. Complex templates comprise all queries with user-defined functions and mathematical expressions on attributes. The optimizer's poor performance makes it unreliable over the complex workload. We believe that optimizer results on simple workload will improve with a higher sampling rate.

Individual templates have different query semantics and access patterns. We compare learning in the black-box approach using the various learning techniques and the optimizer. In the black-box approach, the amount of training data varies (as shown on X-axis). We report the accuracy over all methods for the training set. Figure 5(a) shows learning over Template T1, which is a 2-dimensional range query template. Optimizers are better suited suited to learn from simple queries as in Template T1. However, the range clauses are on two, highly-correlated spatial coordinates, ra and dec. **OPT** errs while propagating the base statistics up the plan, leading to inaccuracies. All of the learning techniques are able to estimate fairly accurately with model trees being the most accurate.

The black-box approach provides the most improvement when operating over queries with a combination of simple range clauses and user-defined functions. For such queries, the optimizer estimates cardinality as 1. While this heuristic provides a reasonable estimate over queries with very low cardinality, SDSS function queries fetch large datasets, leading to large errors. We show the effect of various learning techniques in Figure 5(b) and using Template T3, which has the largest number of queries. Queries in template T3 are proximity queries, which use a user-defined function to find nearby object in a rectangular region. Model trees present the best learning method for such a template. Locally weighted regression, which performs poorly on other templates, estimates reasonably on

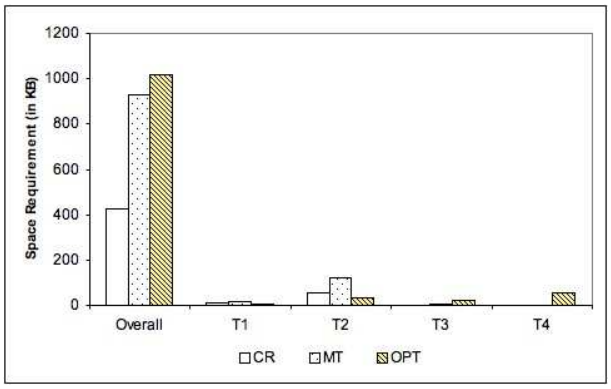


Figure 8: Space Performance

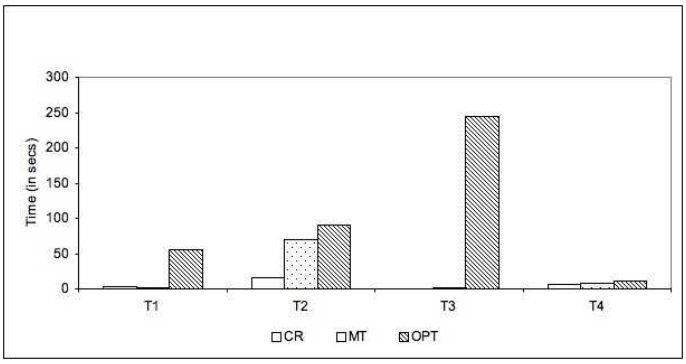


Figure 9: Time Performance

this template.

Scientific queries often use mathematical and bit operators and are quite common in the SDSS workload. Figure 5(c) considers such a query in template T4. Specifically, the predicates in the query add two attributes and compare the result with a constant. However, the optimizer chose to use the primary index and perform a table scan, thereby overestimating the cardinality of each query.

Figure 5(d) shows the comparison of model trees and classification and regression with a self-learning histogram technique [1] using template T1. This technique learns a multi-dimensional histogram from queries and the results apply to simple range clauses only. We do not restructure the histograms as proposed, but refine them incrementally using queries and their results. Their technique learns effectively over queries (providing only 20% error over 30% training data) and can be applied for such templates. The classification and regression approach is slightly less accurate owing to the yield classification step. Model trees are most accurate, but the size of the tree is slightly more than the total histogram size.

Figure 7 shows the influence of various parameters over the cardinality of each query. While the actual number of parameters are high in each template, the learning techniques prune away the unnecessary parameters, thereby reducing the dimensionality of the learning space. This results in succinct data structures, as we will show subsequently.

5.4 Space Utilization and Running Time

The learning techniques have different space requirements. The space requirement of classification and regression (**CR**) is the size of the constructed decision tree. Regression functions have negligible space requirement. Similarly, for model trees, it is the size of the model tree. In locally-weighted regression there is no a priori learned model constructed from the training set, but a new regression model is constructed for each test instance. Therefore, we do not calculate space requirement for **LWR**. For **OPT**, the space requirement is the size of the histograms calculated, as described in the MSSQL 2000 documentation [21]. The overall space requirement was calculated by summing the size of constructed models over all queries at the end of the workload. For individual templates, size was calculated over the minimum number of queries required as training data after which the learned model was considered stable.

Figure 8 shows the space requirement of the black-box approach using different learning techniques and the optimizer over the entire workload. It also shows the individual space requirements of each template. The estimation model data structures used in the black-box approach are very space efficient. Overall, classification and regression has the smallest space requirement, with the size of decision tree varying from tens or hundreds of kilobytes. The succinct decision trees created in classification and regression make it a viable choice for applications where space is at a premium, such as Open SkyQuery. The largest tree overall was that of template T3. When constructed with all 750,000 queries in the template, it required only 286 KB. Even for Template T5, which is a template on an equality predicate for a primary key, the space requirement is extremely low, because there is very little information to classify. Model trees use nearly double the space of classification and regression. In model trees, the space requirement for the workload not in the top six templates compares with the space requirement of decision trees in classification and regression approach. Since this amounts to one-third of the workload, the overall space requirement is less than double the space requirement. However, for the important templates the space requirement is almost doubled. The optimizer space requirement for the overall the workload is comparable to the space requirement of **MT** technique. The optimizer space requirement is calculated from the one-dimensional histograms constructed over individual attributes present in the workload. Because there is high syntactic locality in the workload, *i.e.*, heavy attribute reuse, the optimizer is also space efficient. However, the CR technique competes with the optimizer and provides more accurate estimates. If the optimizer were to be more accurate, it would need multi-dimensional histograms, which add significantly to its space requirements. Additionally, the total space requirement of the optimizer over all attributes, not just over the attributes in the current workload, was 2612KB.

Figure 8 reports the optimizer space requirement for each individual template. The optimizer requires the least space in Template T1, which is a two-dimensional range query template. However, our learning approach, because of high "locality" in query access patterns, learns the distribution using 200 queries and, thus, occupies comparable space.

Estimation runtime performance follows that of space performance, because small data structures may be accessed quickly. The classification and regression technique in the black-box approach exhibits good runtime performance (Figure 9) and model trees are comparable to the optimizer. Locally-weighted regression is an ex-

ception. It takes much longer as the prediction model is rebuilt for each test instance at runtime. We show time results for only the important templates. Decision trees present the most computationally efficient learning data structures, taking only $O(mn(logn))$ construction time in which m is the number of parameters and n are the training examples. The time taken to construct the linear regression function is negligible. For template T5, which took the longest to learn and stabilize, the decision tree was constructed in 15.5 seconds. The optimizer time costs are calculated based on the time it takes to optimize a query and estimate its cardinality. The optimizer results are a clear reflection of the core thesis of this paper that for many applications, expensive optimization is not needed for cardinality estimates and a black-box learning approach suffices. In addition, the total time requirement of the optimizer over all queries was close to one hour.

5.5 Impact on Open SkyQuery

Our final goal is to show that distributed applications benefit significantly from using a black-box approach. We revert to the case of Open SkyQuery and consider the effect on its caching function. The performance of the caching module is measured in terms of total network traffic saved. In one variant of this experiment, the caching module uses exact cardinality values to make caching decisions. We call this this the *prescient* estimator. The prescient estimator gives an upper-bound on how well a caching algorithm could possibly perform. We compare this with the **OPT** method and the **CR** learning technique. Because the **CR** learning technique uses the least amount of space, it is naturally suited to caching. As a cardinality estimator for Open SkyQuery caching, the black-box approach outperforms the optimizer and approaches the ideal performance of the prescient estimator. The total amount of data sent across the network to serve the entire 1706 GB of the SDSS workload. We report the network savings over the entire workload, and compare it with the ideal performance of the prescient estimator. Based on the estimates, **CR** saves 532.15 GB when compared with 307.22 GB for the optimizer. **CR** also compares well with the prescient estimator, reducing network savings by only 5% from 558.49 GB to 532.15 GB. Caching results also show the sensitivity of caching in Open SkyQuery to accurate yield estimates. Nearly all of the benefit of caching may be lost. The Optimizer loses 45% of network savings.

6. CONCLUSIONS

The black-box approach offers an alternative to estimating query cardinalities that is compact, efficient, and accurate. It does not require knowledge of the distribution of data and avoids inaccuracies from modeling assumptions, such as the conditional independence of attributes. In contrast, database query optimizers use a bottom-up method to compose query cardinality estimates, propagating estimates up a query execution plan based on estimates of data distributions. This makes sense for database, because they also need to know cardinalities for each sub-query in order to choose between various query plans, and because they have access to the underlying data. Several emerging applications, such as proxy-caching, data-centric grids, and federated query processing, need to estimate query cardinalities, but have neither access to data distributions nor do they require sub-query cardinalities. Hence, the black-box approach suits them well. In this paper, we demonstrate this using the Open SkyQuery as a case study. We study the cardinality estimation requirements of the Open SkyQuery, describe the black-box approach as applied to it, and present experimental results that show dramatic increases in the performance of caching in the federation.

7. ACKNOWLEDGMENTS

The authors sincerely thank Jim Gray and Alex Szalay for their continued support and involvement from problem conception to suggesting various methods to improve our approach. The authors thank Amitabh Chaudhary for helping us better understand the bottom-up approach and making the descriptions concrete. We also thank Xiaodan Wang for his help the experimental section and explaining the gory details of the Open SkyQuery source code. This work was supported in part by NSF award IIS-0430848, by the IBM Corporation, and by Microsoft.

8. REFERENCES

[1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 181–192, 1999.

[2] J. L. Ambite and C. A. Knoblock. Flexible and scalable query planning in distributed and heterogeneous environments. In *Conference on Artificial Intelligence Planning Systems*, pages 3–10, 1998.

[3] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. DBProxy: A dynamic data cache for Web applications. In *International Conference of Data Engineering*, page 821, Los Alamitos, CA, USA, 2003. IEEE Computer Society.

[4] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. Scalable template-based query containment checking for Web semantic caches. In *Proceedings of the International Conference on Data Engineering*, pages 493–504, 2003.

[5] A. Bagchi, A. Chaudhary, M. T. Goodrich, C. Li, and M. Shmueli-Scheuer. Achieving communication efficiency through push-pull partitioning of semantic spaces in client-server architectures. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, September 2006.

[6] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey Data Reduction Report. *IEEE Data Engineering Bulletin*, 20(4):3–45, 1997.

[7] N. Bruno and S. Chaudhuri. Exploiting statistics on query expressions for optimization. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 263–274, 2002.

[8] N. Bruno, S. Chaudhuri, and L. Gravano. STHoles: A multidimensional workload-aware histogram. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001.

[9] C. M. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 161–172, 1994.

[10] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[11] J. Gray, D. T. Liu, M. Nieto-Santisteban, A. Szalay, D. J. DeWitt, and G. Heber. Scientific data management in the coming decade. *SIGMOD Record*, 34(4):34–41, 2005.

[12] G. Hamerly and C. Elkan. Learning the k in k-means. In *Neural Information Processing Systems*, 2003.

[13] Z. He, B. S. Lee, and R. Snapp. Self-tuning cost modeling of user-defined functions in an object-relational DBMS. *ACM Transanctions on Database Systems*, 30(3):812–853, 2005.

[14] L. Lim, M. Wang, and J. S. Vitter. CXHist : An on-line classification-based histogram for XML string selectivity estimation. In *Proceedings of the International Conference on Very Large Data Bases*, pages 1187–1198, 2005.

[15] M. Litzkow, M. Livny, and M. Mutka. Condor - A hunter of idle workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[16] D. T. Liu and M. J. Franklin. The design of GridDB: A Data-centric overlay for the scientific grid. In *VLDB*, pages 600–611, 2004.

[17] T. Malik, R. Burns, N. Chawla, and A. Szalay. Estimating query result sizes for proxy caching in scientific database federations. In *SuperComputing*, April 2006.

[18] T. Malik, R. C. Burns, and A. Chaudhary. Bypass caching: Making scientific databases good network citizens. In *Intl' Conference on Data Engineering*, pages 94–105, 2005.

[19] T. Malik, A. S. Szalay, A. S. Budavri, and A. R. Thakar. SkyQuery: A WebService Approach to Federate Databases. In *Proceedings of the Conference on Innovative Data Systems Research*, January 2003.

[20] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[21] Statistics in MSSQL Server 2005, http://www.microsoft.com/technet/prodtechnol/sql/2005/qrystats.mspx.

[22] C. Olston, B. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *ACM SIGMOD*, pages 355–366, 2001.

[23] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *ACM SIGMOD*, pages 73–84, 2002.

[24] D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734, San Francisco, 2000. Morgan Kaufmann.

[25] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[26] L. Serafini, H. Stockinger, K. Stockinger, and F. Zini. Agent-based query optimisation in a grid environment. In *International Conference on Applied Informatics*, February 2001.

[27] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[28] Open SkyQuery, http://www.openskyquery.net, 2000.

[29] U. Srivastava, P. J. Haas, V. Markl, M. Kutsch, and T. M. Tran. Isomer: Consistent histogram construction using query feedback. In *22nd International Conference on Data Engineering*, page 39, Washington, DC, USA, 2006. IEEE Computer Society.

[30] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil. LEO - DB2's LEarning Optimizer. In *Proceedings of the International Conference on Very Large Data Bases*, pages 19–28, 2001.

[31] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.