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.

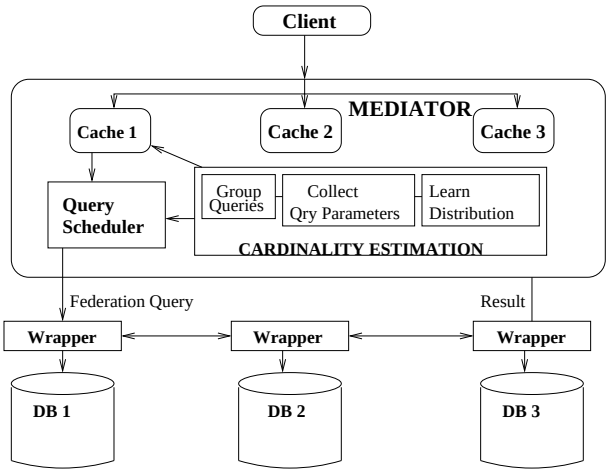

Figure 1: Abstract Architecture of Open SkyQuery

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.

a) Template and its parameters	(b) Cardinality (Cdn) distribution of the parameter vectors

`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}`
	`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}`

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,