ISSN:2348-2079

Volume-6 Issue-2

International Journal of Intellectual Advancements and Research in Engineering Computations

Aggregate Hidden Cloud Web Database Model Using Checkbox Interface

[1]Mr. S.Sambasivam, MCA.,MPhil., Associate profeesor,
[2]Ms. J.Kaviya, Final MCA,
Department of MCA, Nandha Engineering College(Autonomous),Erode-52.
E-Mail ID: sammy2173@gmail.com, kaviyabca96@gmail.com

***Abstract*—Websites provide restrictive form-like interfaces which allow users to execute search queries on the underlying hidden databases. Hidden databases are widely prevalent on the web. This thesis considers the problem of estimating the size of a hidden database through its web interface. Due to limitations of a web interface, the number of returned tuples is usually restricted by a top-k constraint - when more than k tuples in the database match the specified condition, only k of them are preferentially selected by a ranking function and returned to the user. In this proposed system novel techniques which use a small number of queries to produce unbiased estimates with small variance. These techniques can also be used for approximate query processing over hidden databases.**

Index Terms- hidden database, unbiased crawling algorithm, aggregate estimation, left deep tree.

I. INTRODUCTION

Hidden databases are data sources hidden behind the only accessible through a restrictive web search interface. The contents of a hidden database cannot be easily crawled by traditional web search engines. In fact, the restrictive web interface prevents users from performing complete queries as they would with the SQL language. For example, there are hardly any web interface-9es providing aggregate queries such as COUNT and SUM functions. The lower query capability of a hidden database surely reduces its usability to some extent.

The aggregate estimation consists of two components: bias and variance. To produce unbiased aggregate estimations over the hidden databases with checkbox interfaces, develop the data structure of left-deep-tree and define the concept of designated query to form an injective mapping from tuples to queries supported by the web interface. To produce unbiased aggregate estimations over the hidden databases with checkbox interfaces, develop the data structure of left-deep-tree and define the concept of designated query to form an injective mapping from tuples to queries supported by the web interface.To reduce the variance of aggregate estimations, develop the ideas of weighted sampling and special tuple-crawling.

II. RELATED WORKS

Cheng Sheng, Nan Zhang, Yufei Tao And Xin Jin ET AL [1] In this paper, the author stated thata hidden database refers to a dataset that an organization makes accessible on the web by allowing users to issue queries through a search interface. In other words, data acquisition from such a source is not by following static hyper-links.

- Instead, data are obtained by querying the interface, and reading the result page dynamically generated. This, with other facts such as the interface may answer a query only partially, has prevented hidden databases from being crawled effectively by existing search engines.
- To extract all the tuples from a hidden database and algorithms are provably efficient, namely, they accomplish the task by performing only a small number of queries, even in the worst case.
- They also establish theoretical results indicating that these algorithms are asymptotically optimal – i.e., it is impossible to improve their efficiency by more than a constant factor. The derivation of our upper and lower bound results reveals significant insight into the characteristics of the underlying problem. Extensive experiments confirm the proposed techniques work very well on all the real datasets examined.

Michael Benedikt, Pierre Bourhis And Clemens Ley et al [2] describe,the author verified about systems whose transitions consist of accesses to a Web-based data-source. An access is a lookup on a relation within a relational database, fixing values for a set of positions in the relation.

AccLTL, is based on a first-order extension of linear-time temporal logic, interpreting access paths as sequences of relational structures. We also present a lower-level automaton model, Automata, which AccLTL specifications can compile into. They show that AccLTL and A-automata can express static analysis problems related to "querying with limited access patterns" that have been studied in the database literature in the past, such as whether an access is relevant to answering a query, and whether two queries are equivalent in the accessible data they can return. They proved decidability and complexity results for several restrictions and variants of AccLTL, and explain which properties of paths can be expressed in each restriction.

SriramRaghavanAnd Hector Garcia-Molina ET AL [3] crawlers retrieve content only from the publicly indexable Web, i.e., the set of Web pages reachable purely by following hypertext links, ignoring search forms and pages that require authorization or prior registration. In particular, they ignore the tremendous amount of high quality content "hidden" behind search forms, in large searchable electronic databases. They address the problem of designing a crawler capable of extracting content from this hidden Web.

Mitesh Patel, Zhen Zhang And Kevin Chen-Chuan Chang [4] etal [5] describe a Internet– the Web has been rapidly "deepened" by massive databases online: While the surfaceWebhas linked billions of static HTML pages, it is believed that a far more significant amount of information is "hidden" in the deepWeb, behind the query forms of searchable databases. Static URL links– They are assembled into Web pages as responses to queries submitted through the "query interface" of an underlying database. Because current search engines cannot effectively "crawl" databases, such data is believed to be "invisible, "and thus remain largely "hidden" from users (thus often also referred to as the invisible or hidden Web.). Using overlap analysis between pairs of search engines, a white paper estimated 43,000-96,000 "deep Web sites" and an informal estimate of 7,500 terabytes of data– 500 times larger than the surface Web. With its myriad databases and hidden content, this deep Web is an important yet largely-unexplored frontier for information search– While they have understood the surface Web relatively well, with various surveys. This article reports our survey of the deep Web, studying the scale, subject distribution, search-engine coverage, and other access characteristics of online databases. They noted that, while the 2000 study opens interest in this area, it focuses on only the scale aspect, and its result from overlap analysistends to underestimate.

JayantMadhavan, LoredanaAfanasievLyublenaAntovaAndAlon Halevy et al [6] describe the Deep Web refers to content hidden behind HTML forms. In order to get to such content, a user has to perform a form submission with valid input values. The name Deep Web arises from the fact that such content was thought to be beyond the reach of search engines. The Deep Web is also believed to be the biggest source of structured data on the Web and hence accessing its contents has been a long standing challenge in the data management community .Over the past few years, they have built a system that exposed content from the Deep Web to web-search users of Google.com. The results of our surfacing are now shown in over 1000 web-search queries per-second, and the content surfaced is in over 45 languages and in hundreds of domains.

III. SYSTEM METHODOLOGY

A. HIDDEN DATABASE WITH CHECKBOX INTERFACE

Hidden databases are data repositories only accessible through—a restrictive web search interface. Input capabilities provided by such a web interface range from a simple keyword-search textbox to a complex combination of textboxes, dropdown controls, checkboxes, etc. In the hidden database with checkbox interface, a checkbox attribute is represented as a checkbox in the web interface. For example, in the home search website, features (e.g., central air, basement) for a home are represented by checkboxes. The checkbox interface has its specialty. By checking the checkbox corresponding to a value v1, it ensures that all returned tuples contain the value v1. But it is impossible to enforce that no returned tuple contains v2—because unchecking v2 is interpreted as "do-not-care" instead of "not-containing-v2" in the interface.

The limitation placed by the checkbox interface prevents the traditional hidden-database aggregate-estimation techniques from being applied. Specifically, if one considers a feature as a Boolean attribute, then the checkbox interface places a limitation that only TRUE, not FALSE, can be specified for the attribute. As a result, it is impossible to apply the existing techniques which require all values of an attribute to be specifiable through the input web interface.

Let D be a hidden database with m checkbox attributes $A_1; . . .;A_m$ and n tuples. Each checkbox attribute Ai has two values 0 and 1, but only predicates of the form A_i=1 is allowed because of the restriction of the checkbox interface. The typical interface where users can query is by specifying values of a subset of attributes. Now suppose a user selects checkboxes Ai1 ; . . . ;Aij from the interface. With such selections, the user constructs a query with Ai1 ¼ 1; . . .; Aij = 1. The query q presented with the following SQL statement:

SELECT * FROM D WHERE Ai_1 = 1 AND … AND Ai_j = 1, which we denote as $\{Ai_1\& . . . \&Ai_j\}_q$ or directly $\{Ai_1 ; . . .;Ai_{j\}q}$ Notation $\{\}_q$ represents a query with no attribute being checked.

The hidden database will search for all tuples, which refer to as Sel(q), satisfying the user-specified query. There are in total Sel(q) tuples satisfying q, but only min {|Sel(q)|,k}; kg tuples can be returned to the user, where k is as in the top-k restriction. With assume that these tuples are returned according to static ranking functions which ensure that the order of any two returned tuples ti and tj won't change by issuing different queries.

To classify queries into the following three categories, depending upon the number of tuples a query q matches and the top-k restriction:

- |Sel(q)|= 0, this query is underflow. There is no results returned.
- 0 < |Sel(q)|<=k, this query is valid. All results are returned within top-k.
- |Sel(q)| > k, this query is overflow. Only the top-k tuples can be returned together with an overflow flag.

tid	A	B	C
t1	0	0	1
t2	0	1	0
t3	1	0	1
t4	1	1	0
t5	1	1	1

Table 1. Example – Checkbox Attributes

Consider a simple hidden database D with three checkbox attributes A; B; C and 5 tuples t1; . . . ; t5. The hidden database is shown in Table 1, where t_{id} is the tuple's id other than an attribute in the application level. Assuming that the top-k restriction is k = 2. The static ranking function is according to the subscript of t_{id} from small to large order. Suppose a user, in this running example, selects the attribute A as user query.

The corresponding SQL statement is, SELECT * FROM D WHERE A=1 it can see that tuples t3; t4; t5 match this query in the hidden database. But with the top-k restriction, only 2 tuples, t3 and t4, can be returned to the user with an overflow flag.

B. HIDDEN DATABASE SAMPLING ANDLEFT-DEEP TREE

To estimate the size of a hidden database, one intuitive idea is to perform tuple sampling. Assume that a tuple t with probability p(t), we can easily estimate the size of the hidden database as ~n = 1/p(t). However, tuples cannot be directly sampled, because they can only be accessed through the queries provided by the hidden database. The hidden database D has m checkbox attributes as its query interface, one can enumerate that there are in total 2^m possible queries, from $\{\}_q$ to $\{A1\ \&\ …\ \&\ Am\}_q$ which are all possible combinations of the m attributes.

All of these 2^m queries are in the query space. Because if it discard any query from them, may not be able to access to some tuples which are only returned by the discarding queries. To organize all these queries in the query space with a left-deep tree structure, where every node is corresponding to a query and a directed edge from a node to a child node indicates that the query corresponding to this child node includes all attributes in the parent query and one additional attribute. The root node * represents query $\{\}_q$, while the bottom leaf A1 & … & Am represents a query with all attributes being checked.

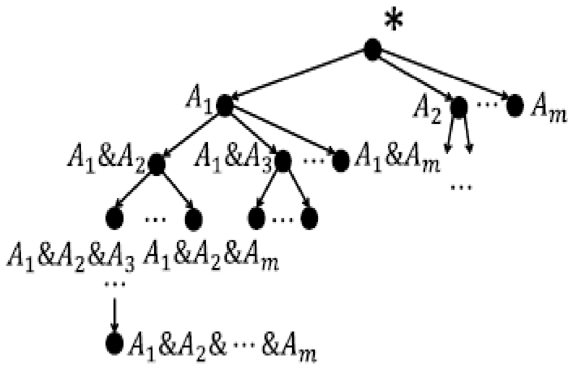

Fig1. Left Deep Tree

There are in total eight (i.e. 2^3) nodes representing eight possible queries. Children of each node are arranged in alphabetic order, such that {A & B}q € {A & C}q. If a tuple t has node q as its designated query, then we tag t's corresponding tid near q.

Assign a tuple to exactly one query, called designated query, which is essentially a one-to-many mapping from queries to tuples, i.e., a tuple can be designated to one and only one query, while a query may designate multiple tuples. With such a mapping, one can see that the probability for sampling a tuple can be easily derived from the sampling probability for the query that returns it specifically, if |d(q)| is the total number of the tuples designated by q, then the probability for sampling a tuple t returned by q is p(q)/d(q) where p(q) is the probability to sample query q.

C.UNBIASED-WEIGHTED-CRAWL ALGORITHM

The unbiased weighted crawl algorithm is used for COUNT estimation on the left deep tree. The drill-down sampling starts at the root node, then randomly drills down recursively along the left-deep tree until reaching a node which no longer overflows (i.e., either underflow or valid). The sequence of visited nodes q_0; q_1; q_2; . . . ; q_h, where q_0 is the root node, is called the path of this drill-down procedure. Then the number of tuples in D is estimated as:

$$\tilde{n} = \sum_{i=0}^{h} \tilde{n}_i = \sum_{i=0}^{h} \frac{|d(q_i)|}{p(q_i)},$$

-

where i represents the level of the node, h is the total length of the drill-down path, $|d(q_i)|$ is the number of tuples with designated query being node q_i, and $p(q_i)$ is the probability for the drill-down process to access q_i, which can be computed as:

$$\mathbf{p(q_i)=p(q_i / q_{i-1})\, p(q_i / q_{i-2}) .. p(q_0) \text{ where } p(q_0)=1}$$

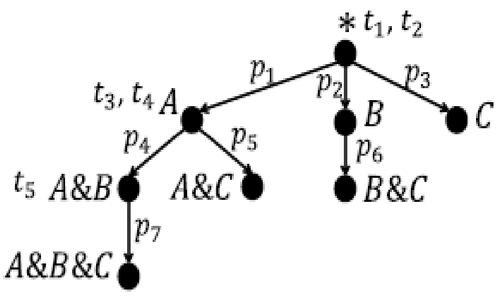

Fig 2. Sample Left Deep Tree

To estimate the number of tuples in D, the algorithm will first use some fixed probabilities to assign weights to edges of the left-deep tree, then randomly drill down and estimate the number of tuples in D. The second step will be recursively executed until the query cost exceeds the query budget. The weight allocation does not affect the unbiasedness of the estimation algorithm as long as each tuple has a probability to be retrieved.

To ensure an unbiased estimation, a fundamental requirement is for the weight of each edge to obey the following rules: The sum of weights of edges under one node should equal to 1; Every edge has a non-zero weight whenever there exists a tuple with designated query being a node in the subtree under this edge. the number of tuples that have been designated by nodes in a subtree is proportional to the number of nodes in this subtree.

Assign the weights of edges corresponding to the number of nodes in their pointed subtrees. With the left-deep tree structure, suppose a node q has j children q_1,$q_2$$q_j$ from left to right. Then the proportion of edge weights under q from left to right should be p(q1/q):p(q2/q): … p(q3/q) = 1/2 : 1/4 : ….$1/2^j$.

The algorithm unbiased-independent conducts the aggregate estimation using two phases. In the first phase, unbiased algorithm is executed to perform drilldown sampling with structure-based weight allocation scheme on the left-deep tree. At the same time, visited tuples are gathered into a set T.

When conducting drill-down sampling, an increasing number of tuples are gathered. Therefore, it is intuitive to use those valuable tuples to learn the data distribution in the hidden database. More specifically, using q as a query, we want to make an estimation of the total number of tuples, denoted as n(q), having nodes in the subtree under node q as theirs designated queries.

D. ADMIN MODULE

Admin Module is developed only for the Higher Level Executives to create and View all the details of the following options. The admin module includes category entry, items entry and view users

The admin module is used to generate the dataset for the experimental system for aggregate estimation. It is used to add the items information to the data base. Also by using this module, the attribute information will be added into the database.

The item details attribute details and user details will be stored and maintained in this module.

E. USER MODULE

The user module is used to register the user and the details will be stored into the database. The registered user can able to view the products which are already added form the admin module.

User Module is developed for all users even they are not registered in the site. The item categories and item details, item categories are viewed by them.

In this module user can able to view the item list by means of selecting the category and top N records from the database or from the web server cache until the access will reach. If the access count will be reached, then the user will not be able to view the item list. Ie (1) a top-k restriction on the number of returned tuples, and (2) a limit on the number of queries one can issue (e.g., per IP address per day) through the web interface.

This module also provides an option for changing the old password of the user. During the modification, the old password and new password are to be given so that the new password is effective in successive logins.

F. AGGREGATE ESTIMATION MODULE

This module is used to enabling the aggregate queries over a hidden database with checkbox interface by issuing a small number of queries (sampling) through its web interface. In this module, an aggregate estimation algorithm is implemented and used that provides completely unbiased estimates for COUNT and SUM queries.
In the first phase, unbiased algorithm is executed to perform drilldown sampling with structure-based weight allocation scheme on the left-deep tree. At the same time, visited tuples are gathered into a set T. In the second phase, we use T to compute **p(Ai)**, for **i = 1 to m,** and call independent weight allocation to adjust weights of edges. Then, drill-down sampling algorithm is performed with theupdated weight allocation of edges, and **T** is also updated with newly gathered tuples.

The second phase will be recursively executed until the query cost exceeds the query budget. There is a pre-determined pilot budget **w** to separate the above twophases, where **w** is the number of queries. When the number of queries exceeds w, the algorithm estimates **p(Ai)** and adjusts weight allocation accordingly, then performs new drill-down sampling. And the algorithm produces unbiased (for COUNT and SUM) aggregate estimations with small variances with the number of tuples and top-k restrictions.

IV. CONCLUSION

This project addresses a novel problem where checkboxes exist in the web interface of a hidden database. To enable the approximation processing of aggregate queries and develops algorithmUNBIASED – WEIGHTED – CRAWLwhichperforms random drill-downs on a novel structure of queries which we refer to as a left-deep tree and also propose weight adjustment and low probability crawl to improve estimation accuracy.

This project performed a comprehensive set of experiments on synthetic and real-world datasets with varying database sizes (from 5;000 to 100;000), number of attributes (from 20 to 50) and top-k restriction (from k = 10to 30). We found that, as predicted by the theoretical analysis, the relative error decreases when the number of queries issued increases. In addition, for the same query budget, the relative error is lower with a smaller number of attributes and/or a large k. In the worst-case scenario, achieve around 25 percent relative error with 300 queries issued for the synthetic dataset, and less than 10 percent relative error with about 2,500 queries issued for the real-world dataset. The experimental results demonstrate the effectiveness of proposed algorithms.

REFERENCES

[1] C. Sheng, N. Zhang, Y. Tao, and X. Jin, “Optimal algorithms for crawling a hidden database in the web,” Proc. VLDB Endowment, vol. 5, no. 11, pp. 1112–1123, 2012.

[2] M. Benedikt, G. Gottlob, and P. Senellart, “Determining relevance of accesses at runtime,” in Proc. 30th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst., 2011, pp. 211–222.

[3] Epicurious, Food search page [Online]. Available: http://www.epicurious.com/ recipesmenus/advancedsearch, 2013.

[4] Homefinder, Home finder page [Online]. Available: http://www.homefinder.com/search, 2013.

[5] A. Dasgupta, X. Jin, B. Jewell, N. Zhang, and G. Das, “Unbiased estimation of size and other aggregates over hidden web databases,” in Proc. Int. Conf.Manage. Data, 2010, pp. 855–866.

[6] Monster, Job search page [Online]. Available: http://jobsearch. monster.com/ AdvancedSearch.aspx, 2011.

[7] M. Benedikt, P. Bourhis, and C. Ley, “Querying schemas with access restrictions,” Proc. VLDB Endowment, vol. 5, no. 7,pp. 634–645, 2012

[8] R. Khare, Y. An, and I.-Y. Song, “Understanding deep web search interfaces: A survey,” ACM SIGMOD Rec., vol. 39, no. 1, pp. 33–40, 2010