CODD: A Dataless Approach to Big Data Testing

Ashoke S. Jayant R. Haritsa
Database Systems Lab, SERC/CSA
Indian Institute of Science, Bangalore

1. INTRODUCTION

The construction and development of the so-called Big Data systems has occupied centerstage in the data management community in recent years. However, there has been comparatively little attention paid to the *testing* of such systems, an essential pre-requisite for successful deployment. This is surprising given that traditional testing techniques, which typically involve construction of representative databases and regression query suites, are completely impractical at Big Data scale – simply due to the time and space overheads involved in their execution. For instance, consider the situation where a database engineer wishes to evaluate the query optimizer's behavior on a futuristic Big Data setup featuring "yottabyte" (10^{24} bytes) sized relational tables. Obviously, just generating this data, let alone storing it, is practically infeasible even on the best of systems.

Motivated by the above lacuna, our lab has developed a graphical tool called **CODD** (Constructing Dataless Databases), which takes a first step towards the effective testing of Big Data deployments, by implementing a new metaphor of "data-less databases" [2]. Specifically, CODD implements a unified visual interface through which database environments with the desired *meta-data* characteristics can be efficiently simulated without persistently generating and/or storing their contents. In addition, these databases will appear to be hosted on the desired computational platforms. As a case in point, the above-mentioned yottabyte scenario operating on a massively parallel cluster can be easily modeled within a few minutes on a vanilla laptop.

In conjunction with meta-data construction, CODD incorporates a graph-based model of the structures and dependencies of the metadata entities. This model is leveraged to implement a topological sort-based checking algorithm to ensure that the metadata values input by the user are both *legal* (valid range, correct type) and *consistent* (compatible with the other meta-data values).

Another unique feature of CODD is its support for automated *scaling* of meta-data instances to suit not only space targets (e.g. boosting a database from 100 GB to 100 TB), but more potently, *time targets* (e.g. rework the metadata such that the overall execution time of a test query workload is multiplied by a user-specified factor). The latter feature is implemented by modeling optimizer plan costs for the query workload as algebraic functions of the scaling factors of the relations featuring in the queries. Subsequently, an inverse minimization function is computed to determine the factor values expected to produce the desired time scaling.

While Big Data has many different facets, including *variety*, *velocity*, and *volume*, CODD currently addresses only the *volume* aspect. Notwithstanding this limited scope, it has been successfully used in the following deployment scenarios:

- In conjunction with the Picasso tool [5], to graphically simulate and visualize *query optimizer* behavior in Big Data environments.
- To cheaply and proactively identify *design bugs* that surface at Big Data scale in database engines.
- To verify that the attractive worst-case performance guarantees provided by the "plan bouquet" query processing technique [1], are achievable even at Big Data scale.

On the implementation front, CODD is completely written in Java, running to over 50K lines of code. It is operational on a rich set of industrial-strength optimizers, including IBM DB2, Microsoft SQL Server, Oracle, HP SQL/MX and PostgreSQL. CODD is freely downloadable [6], and is currently in use at industrial and academic institutions worldwide.

Demo Features. The demo visually and interactively showcases the various features of CODD on industrial-strength database engines. The highlights of the presentation, which are detailed in Section 4, include: (a) Metadata construction to simulate Big Data databases; (b) Time-based scaling of metadata; (c) Characterizing query optimizer behavior on Big Data databases; (d) Demonstration of design bugs surfacing at Big Data scale; and (e) Verifying that plan bouquet's good robustness extends to Big Data scenarios. A complete video of CODD in operation is available at the project website [6].

2. THE CODD METADATA PROCESSOR

In this section, we provide an overview of CODD's architecture and main features – the full technical details are available in [2].

Database systems maintain a large corpus of meta-data, covering various aspects of their operation. The current CODD prototype primarily supports the construction of *statistical metadata* related to *query optimization*, including the following entities: (a) *Relation statistics*: (cardinality, row length, number of disk blocks, etc.,); (b) *Attribute statistics* (column width, fraction of nulls, histogram bounds, number of distinct values, etc.,); (c) *Index statistics* (cardinality, leaf blocks, cluster factor, etc.,); and (d) *System parameters* (number of CPUs, CPU speed, sort heap size, etc.,).

Proceedings of the VLDB Endowment, Vol. 8, No. 12

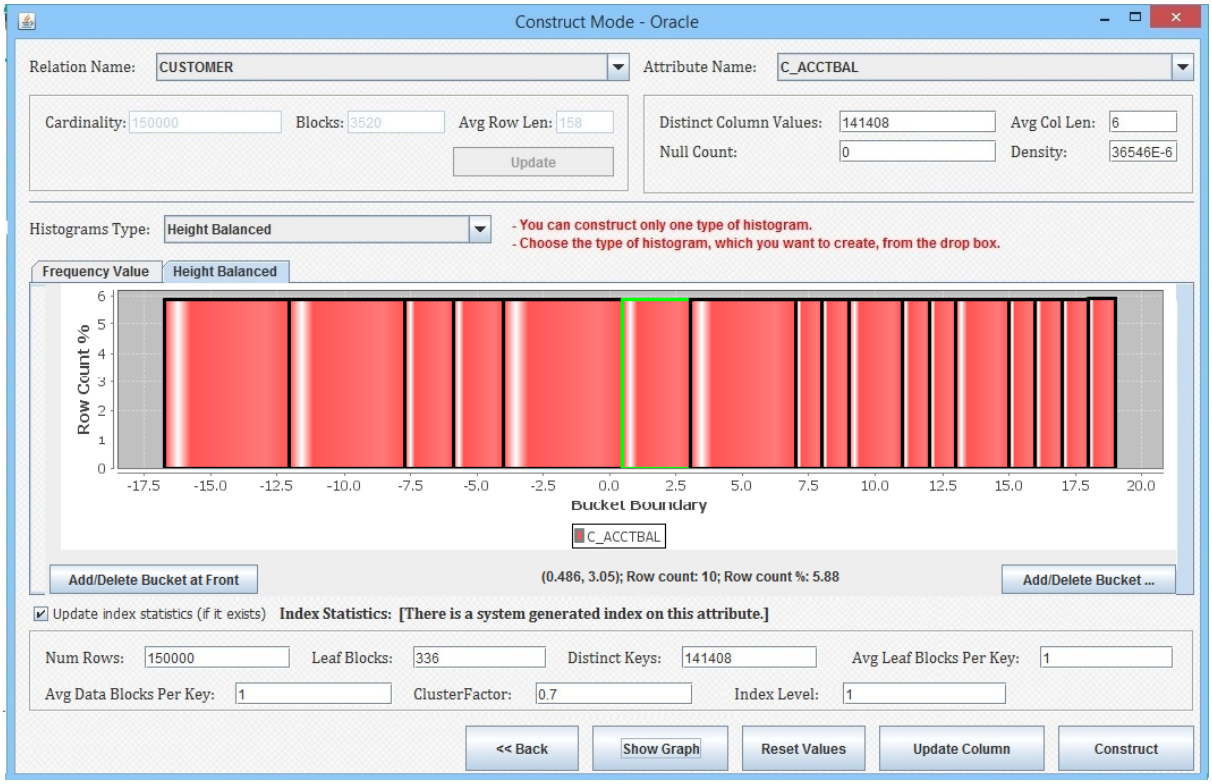


Figure 1: Metadata Construction Interface

2.1 Metadata Construction

Given a database schema with empty content, CODD simulates the desired metadata configuration by asking the user to input, in sequence, the relation statistics for all the tables in the schema, followed by the column and index statistics for each attribute in each relation. The entry of this information is through a convenient graphical interface, a screenshot of which is shown in Figure 1. To push these values into the database catalogs, CODD leverages the native techniques provided by the underlying engine for statistics updates, but adds convenience by (a) packaging them in a largely vendor-neutral interface, and (b) camouflaging the details through dynamically created SQL procedures.

For easily creating the histograms corresponding to column distributions, CODD provides a graphical editing mode wherein the individual bucket geometries can be modified using the mouse, subject to the histogram types permitted by the underlying engine – for instance, an equi-depth histogram, as shown in Figure 1. Moreover, CODD also offers the user a pre-defined menu of classical distributions (e.g. *Uniform*, *Zipf*, *Normal*, etc.), and choosing any of these results in the appropriate histogram being automatically created. Subsequently, the user can make local variations on this base histogram, if so desired.

2.2 Metadata Validation

Since CODD facilitates users to directly enter meta-data, we need to ensure that the input information is both legal (valid type and range) and consistent (compatible with other metadata values). The validation process involves the construction of a *directed acyclic constraint graph*, which concisely represents all the metadata entities along with their structural and consistency constraints. Specifically, each node in the graph represents a single metadata entity, annotated with legality constraints, and the currently assigned values to the entity which must adhere to these constraints. The directed edges between nodes represent the statistical value dependencies between the associated metadata entities. The edge direction is always from the node at the higher level of abstraction to the lower level node (e.g. from *relation* to *attribute*), while for nodes at the same level, the edge goes from the aggregate to the specific (e.g. from *cardinality* to *distributions*), and for the remainder, a lexicographic ordering is used.

A sample portion of a constraint graph, corresponding to a popular commercial database engine, hereafter referred to as **ComOpt**, is shown in Figure 2, covering relation, attribute and index metadata entities. Here, if we consider the node *Card*, which represents the cardinality of the relation, the associated legality constraint specifies that the value should be greater than or equal to zero. Similarly, the consistency constraint represented by the directed edge from *Card* to *Null Count*, specifies that *Null Count*, the number of null values in the column, cannot exceed *Card*.

After construction of the constraint graph, a topological sort is run to obtain a linear ordering on the graph. Then, CODD guides the user to enter the metadata details in exactly this order, requesting inputs at each new node, and ensuring that all applicable constraints are met. The linear ordering sequence is shown in Figure 2 by the (bracketed) number associated with each node.

2.3 Metadata Scaling

CODD supports two flavors of database scaling, *size-based scaling* and *time-based scaling* – while the former is commonplace, the latter has not been previously attempted in the literature.

In size-based scaling, given an initial metadata configuration, denoted by M, and a positive integer scaling factor α entered by the

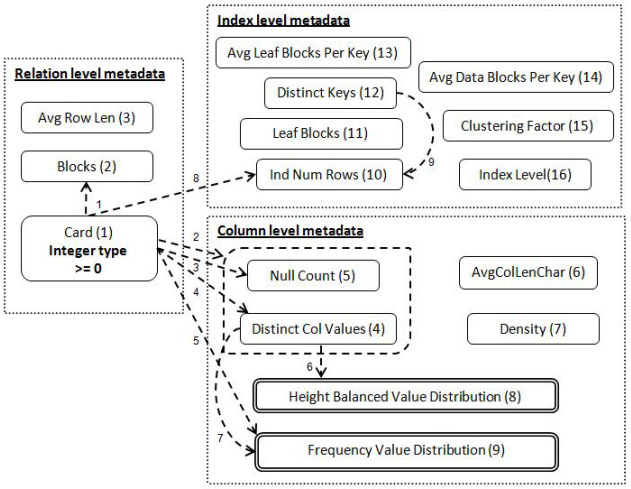


Figure 2: Metadata Constraint Graph (ComOpt)

user, CODD produces a scaled metadata configuration M^α such that the size of the new database is α times the size of the initial configuration. It does so by linearly scaling the cardinalities of the relations, accompanied by domain-cardinality scaling for the primary keys and foreign keys featuring in the scaled tables. This approach is similar to the scaling mechanisms found in standard benchmark databases such as TPC-H [4].

In time-based scaling, on the other hand, the input also includes a query workload Q. Here, the objective is to scale M such that the optimizer's estimated *time* for executing Q on M is scaled by the factor α when executed on M^α. In particular, the following optimization problem is solved, with the intention of having, as far as possible, each individual query's cost to be scaled by α:

Produce an M^α such that the sum over Q of the individual squared deviations from α in cost scaling is minimized, subject to the constraint that the overall cost over Q is scaled by α.

The modeling and solution of this problem is described in [2]. It is an instance of sequential quadratic programming with both equality and inequality constraints, and CODD uses the Suan-Shu [3] numerical library to solve this optimization formulation.

3. CODD IN ACTION

As already mentioned, CODD is currently deployed at various industrial and academic sites. In this section, we present various scenarios that highlight its utility.

3.1 Query Optimizers on Big Data

With CODD, we can easily simulate and assess a query optimizer's behavior in response to futuristic scenarios. For instance, given a parameterized SQL query template that defines a relational selectivity space, we can combine CODD with the Picasso tool [5] to produce *Plan Diagrams* and *Cost Diagrams* for the desired Big Data environment. A plan diagram is a visual representation of the plan choices made by the optimizer over the parameter space, while the cost diagram quantitatively depicts the estimated query processing costs of the plans shown in the plan diagram.

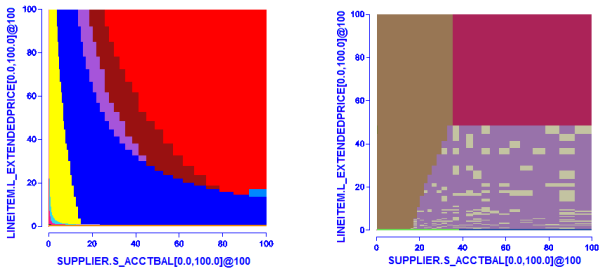
To make this notion concrete, consider QT5, the parameterized SQL query template shown in Figure 3, which is based on TPC-H Query 5. Here, selectivity variations on the SUPPLIER and LINEITEM relations are specified through the *s_acctbal :varies* and *l_extendedprice :varies* predicates, respectively.

```
select n_name,
       sum(l_extendedprice * (1 - l_discount)) as revenue
from customer, orders, lineitem, supplier, nation, region
where c_custkey = o_custkey and l_orderkey = o_orderkey
and l_suppkey = s_suppkey and c_nationkey = s_nationkey
and s_nationkey = n_nationkey and n_regionkey = r_regionkey
and r_name = 'ASIA' and o_orderdate >= '1994-01-01'
and o_orderdate < '1995-01-01' and c_acctbal <= 5400
and s_acctbal :varies and l_extendedprice :varies
group by n_name
order by revenue desc
```

Figure 3: Query Template QT5

The associated plan diagram produced by ComOpt on the baseline 1 GB TPC-H database hosted on a vanilla single-server configuration, is shown in Figure 4(a). In this picture, each colored region represents a specific plan, and a set of 21 plan choices cover the selectivity space.

Now consider a scenario where the optimizer developer would like to assess its behavior on a 1 PB version of the TPC-H benchmark, operating on a scaled-out system with 1000 servers. Using the metadata construction and scaling features of CODD, we can easily create a metadata shell that emulates this futuristic environment. The plan diagram produced by ComOpt for this scenario is shown in Figure 4(b). We observe that the number of plans has sharply decreased from 21 to 12 in the scaled version, and the geometries of the plan optimality regions have undergone significant changes. Morever, the plans themselves are quite different between the two pictures.



(a) 1 GB / 1 Server (b) 1 PB / 1000 Servers

Figure 4: Plan Diagrams for QT5 (ComOpt)

3.2 Design Bugs at Big Data scale

CODD has also proved to be effective at identifying *bugs* that surface at Big Data scale. For instance, by iteratively executing CODD with QT5 on ComOpt, and having the database size increase in each iteration, we were able to quickly discover that ComOpt's cardinality estimation module started seriously malfunctioning when the input data size to an operator exceeded *20 exabytes* – specifically, the output cardinality of the operator became frozen at approximately *18.4 exabytes*, no matter how large the input was beyond this point! This is visually shown in the (partial) plan tree of Figure 5, where the circled regions highlight the erroneous behavior.

A second bug example is shown in the cost diagram of Figure 6a, corresponding to the QT5 template operating on a *100 petabyte* database shell (for readability, only the cost variation on the LINEITEM selectivity is shown). Here, we observe that the query execution costs initially rise with the selectivity, as expected.

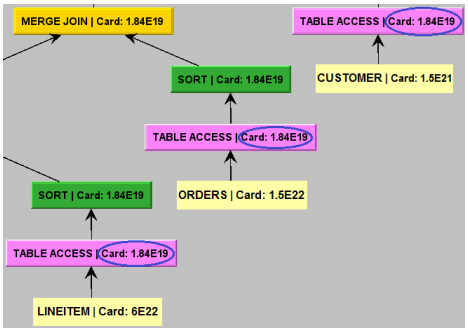
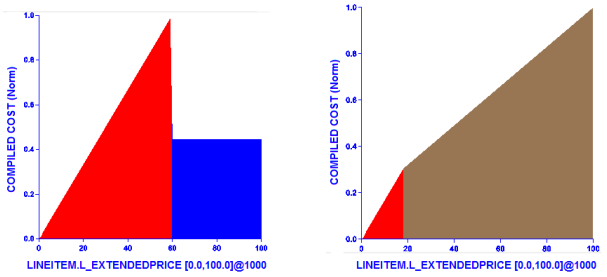


Figure 5: Output Cardinality Error (ComOpt)

However, at a selectivity of 60 percent, there is a sudden and substantial dip followed by a saturation in the query execution cost, violating the *cost-monotonic* behavior that typically holds for such queries. The primary change is a switch from a nested-loops join of LINEITEM and SUPPLIER in the red plan to a sort-merge join in the blue plan.

Interestingly, this incorrect behavior disappears when the processor speed is scaled up by a factor of 100, as shown in Figure 6(b). These results strongly suggest the presence of underlying issues in the construction of ComOpt's cost estimation module.



(a) 100 PB / 1 Server (b) 100 PB / x100 Scaleup Server

Figure 6: Cost Diagrams for QT5 (ComOpt)

3.3 Robustness Guarantees at Big Data scale

A well-documented problem in the database literature is that selectivity estimates for optimizing decision support queries often differ significantly from those actually encountered during query execution, leading to poor plan choices and inflated response times. A new approach to address this classical problem was recently proposed in [1], wherein the compile-time estimation process is completely jettisoned – instead, selectivities are systematically *discovered* at run-time through a precisely calibrated sequence of cost-limited executions from a carefully chosen small set of plans, called the "plan bouquet". A potent benefit of this discovery-based approach is that it lends itself, for the first time in the literature, to providing *guaranteed bounds* on worst-case query processing performance. More precisely, if we define **MSO** (Maximum Sub-Optimality) as the worst-case ratio of the cost sub-optimality incurred by the database engine, relative to an oracular system that magically knows the correct selectivity values, it is guaranteed that $MSO \leq 4 * |PlanBouquet|$.

For conventional database environments, it was found in [1] that the bouquet cardinality is typically a small number, around 3 or 4, leading to MSOs of less than 20 – this is miniscule in comparision to current query optimizers whose MSOs are in the thousands!

Database Size	PostgreSQL MSO	PlanBouquet MSO
1 GB	2000	12
100 GB	5000	16
10 TB	300000	16

Table 1: MSO Performance wrt Database Size

However, an open question was whether this attractive performance extends to Big Data environments. We answered this by leveraging CODD to experimentally confirm on PostgreSQL, as shown in Table 1, that the plan bouquet approach *continues* to provide low MSO values at Big Data scales – for instance, on a 10 TB database, its MSO is only 16. In marked contrast, the native PostgreSQL engine's performance becomes progressively worse with increasing database size, reaching an MSO of 300000 at the 10 TB scale!

4. CODD DEMONSTRATION

In the demo, the audience will actively engage with a variety of visual scenarios that showcase the utility of the CODD tool.

Firstly, we will present the Construct Mode of CODD on the TPC-H schema and show how arbitrary "what-if" metadata scenarios could be constructed from scratch – in particular, the *yottabyte* scenario mentioned in the Introduction. The GUI for creating histograms for attribute distributions will also be emphasized.

User Interaction: The audience will be able to input all the metadata details for a sample relation. They will also be encouraged to provide illegal and inconsistent values, and verify that these are caught by the validation checks employed in CODD. The audience will also get to play with the graphical histogram interface to model changes in the data distributions.

Secondly, we will move on to demonstrating the scaling functionalities supported by CODD. For size-based scaling, starting from the standard 1GB TPC-H metadata statistics, we will simulate a Big Data setup featuring yottabyte-sized relational tables. For time-based scaling, we will take a set of TPC-H benchmark queries as the query workload, and compute the relation size scalings required to produce the desired time scaling of this workload.

User Interaction: The audience will be able to choose the scaling factor, and verify that the metadata statistics have been changed appropriately to reflect the desired Big Data scenario.

Finally, we will showcase the bugs that were discovered at Big Data scale in ComOpt, as described in Section 3. This will be followed by demonstrating on PostgreSQL that the plan bouquet's attractive robustness guarantees are maintained even in these extreme environments, whereas the native optimizer performs very poorly.

Acknowledgements. We thank Rakshit Trivedi, I. Nilavalagan, Deepali Nemade and Ankur Gupta for their valuable contributions to the development of CODD over the past five years, and Suresh Soundararajan of HP for his detailed evaluation of its features.

5. REFERENCES

[1] A. Dutt and J. Haritsa, "Plan Bouquets: Query Processing without Selectivity Estimation", *Proc. of ACM SIGMOD Intl. Conf. on Management of Data*, 2014.

[2] R. Trivedi, I. Nilavalagan and J. Haritsa, "CODD: Constructing Dataless Databases", *Proc. of 5th Intl. Workshop on Testing Database Systems (DBTest)*, 2012.

[3] numericalmethod.com/suanshu/

[4] www.tpc.org/tpch

[5] dsl.serc.iisc.ernet.in/projects/PICASSO

[6] dsl.serc.iisc.ernet.in/projects/CODD