HYDRA: A Dynamic Big Data Regenerator

Anupam Sanghi◇ Raghav Sood◇ Dharmendra Singh◇ Jayant R. Haritsa◇ Srikanta Tirthapura♣
◇Indian Institute of Science, Bangalore, India   ♣Iowa State Univerity, Ames, USA
{anupamsanghi, dharmendra, haritsa}@iisc.ac.in   raghavsood33@gmail.com   snt@iastate.edu

ABSTRACT

A core requirement of database engine testing is the ability to create synthetic versions of the customer's data warehouse at the vendor site. Prior work on synthetic data regeneration suffers from critical limitations with regard to (a) scaling to large data volumes, (b) handling complex query workloads, and (c) producing data on demand. In this demo, we present **HYDRA**, a workload-dependent dynamic data regenerator, that materially addresses these limitations. It introduces the concept of dynamic regeneration by constructing a minuscule memory-resident database summary that can on-the-fly regenerate databases of arbitrary size during query execution. Further, since the data is generated in memory, the velocity of generation can be closely regulated. Finally, to complement dynamic regeneration, Hydra also ensures that the process of summary construction is data-scale-free.

1. INTRODUCTION

In industrial practice, relational database vendors often need to test their OLAP engines on the data warehouses present at the customer sites. This requirement arises for reasons like: (a) analyzing performance issues during query processing, (b) performing functional testing of embedded-SQL programs, and (c) proactively assessing the performance impacts of planned engine upgrades. Due to privacy concerns, however, transferring data from the client to the vendor may not be a viable option. Moreover, even if the client is willing to share, transferring and storing the data at the vendor's site may have impractical time and space overheads, especially in the impending Big Data era. Therefore, looking into the future, vendors need to be able to *dynamically regenerate* representative databases that mimic, for the intended purposes, the behavior of the client data processing environments.

A rich body of literature exists on data regeneration, including both *workload-independent* (WI) techniques (e.g. [7, 8]) and

workload-dependent (WD) techniques (e.g. [6, 9, 4]). The WI approaches, including those that are specifically targeted towards Big Data environments (e.g. [11, 13]), fail to retain satisfactory statistical fidelity. This shortcoming is addressed in the WD schemes – specifically, they generate synthetic data that exhibits *volumetrically similar* behavior to the original database on the customer query workload. That is, with common query execution plans at the client and vendor sites, the output row cardinalities of individual operators in these plans are almost identical. However, these techniques suffer from substantive practical limitations, especially with regard to: (a) scaling to large data volumes, (b) handling complex query workloads, and (c) producing data on demand.

We have attempted to address the above limitations of WD techniques in designing a new database regenerator called **Hydra** [12]. This tool forms part of our ongoing CODD project [5], which incorporates a novel metaphor of "dataless databases", whereby database environments with the desired characteristics are simulated *without* persistently generating and/or storing the contents.

Hydra currently focuses on the *volume* and *velocity* facets [1] of Big Data, which are of primary interest in the context of enterprise relational warehouses. It introduces the concept of *dynamic regeneration* by constructing a minuscule memory-resident *database summary* that can on-the-fly regenerate arbitrary client databases during query execution. Since the data is generated in memory, the velocity of data generation can be closely regulated, as compared to disk-resident databases. To complement dynamic regeneration, Hydra also ensures that the process of summary construction is *data-scale-free*. Specifically, the summaries for complex Big Data client scenarios are constructed within just a few minutes.

On the implementation front, Hydra is completely written in Java, running to over 15K lines of code, and is currently operational on the PostgreSQL v9.3 engine [3]. It has an intuitive user interface that facilitates modeling of enterprise database environments, delivers feedback on the regenerated data, and tabulates performance reports on the regeneration quality. The entire tool, including the source, can be downloaded at [2], and has already been deployed in major telecom and software organizations.

Demo Highlights. The highlights of the demo (detailed in Section 4) include the following components: (a) **Client Interface**, which captures the construction and transfer of the information synopsis created at the client site; (b) **Vendor Interface**, which presents the synthetic database summary and explicit verification of volumetric similarity; (c) **Dynamic Regeneration**, which demonstrates on-the-fly data generation during query execution on a dataless database; and (d) **Scenario Construction**, which helps the vendor to pro-actively simulate anticipated client environments.

2. HYDRA DESIGN

Hydra leverages the declarative approach to data regeneration proposed in the DataSynth tool [4]. To illustrate this approach, consider a relational database with the toy schema shown in Figure 1a (where the pk and fk denote primary-key and foreign-key attributes, respectively). A sample query on this schema is listed in Figure 1b, and the corresponding execution plan in Figure 1c. A special aspect of this plan is that the output edge of each operator is annotated with the associated row cardinality, as evaluated during the client's execution. It is therefore referred to as an *Annotated Query Plan* (AQP) [6]. The AQPs constructed over the entire query workload are then collectively formulated as a set of linear programs (LPs), one per schema relation. These LPs separately are then input to an SMT solver, and the solutions are used to construct the synthetic database. The objective of the regeneration process at the vendor site is to ensure that the synthetic data closely mimics the operator data volumes indicated in the AQP. This helps to preserve the multi-dimensional layout and flow of the data, a pre-requisite for achieving comparable performance on the client's workload.

| R (R_pk, S_fk, T_fk) | S (S_pk, A, B) | T (T_pk, C) |

(a) Database Schema

```
select * from R, S, T
where R.S_fk = S.S_pk   and   R.T_fk = T.T_pk
and S.A >= 20 and S.A < 60   and   T.C >= 2 and T.C < 3
```

(b) Example Query
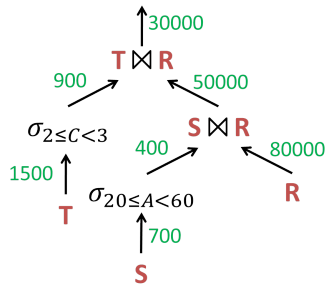
(c) Annotated Query Plan (AQP)

Figure 1: Example Database Scenario

Novelties. Hydra is able to handle significantly more **complex query workloads** than DataSynth, as detailed in [12]. This improvement is due to a novel *region-partitioning* algorithm [12] that results in an LP encoding whose complexity (in terms of the number of variables) is several orders of magnitude smaller in comparison to the *grid-partitioning* approach of [4]. In fact, our region-partitioning corresponds to an LP with the *minimum* number of variables [12].

Second, Hydra introduces the concept of **dynamic regeneration** by constructing a minuscule *database summary* that can on-the-fly regenerate databases of arbitrary size during query execution. This approach is imperative for Big Data systems, where working with materialized solutions entails impractical time and space overheads. Specifically, dynamic generation eliminates the need to store data on the disk and its subsequent load by the engine – instead, all data is created and delivered on demand. An orthogonal benefit is that the generation rate can be strictly controlled, thereby addressing the *velocity* aspect of Big Data.

Third, our database summary generation, thanks to its unique **data-scale-free** feature, is extremely efficient. As a case in point, the summary for a large workload of 131 distinct queries on the TPC-DS database was generated in less than 2 minutes on a vanilla computing platform, occupying only a few KB of space [12].

Finally, our summary generation method also delivers **better fidelity** than prior work with regard to volumetric similarity. For instance, on the above-mentioned query workload, more than 90% of the volumetric constraints were satisfied with virtually no error, while the remaining were all satisfied with a relative error of less than 10% [12]. Further, since the magnitude of the volumetric discrepancy is constant for a given query workload, the relative errors becomes progressively smaller with increasing database size.

The above efficiency and accuracy in constructing the summary are an outcome of the *deterministic alignment* strategy of Hydra (details in [12]), as opposed to the *sampling*-based strategy of [4].

3. HYDRA ARCHITECTURE

We now present an overview of Hydra's architecture, shown in Figure 2. In this figure, the green boxes represent the new components designed specifically for Hydra, whereas the yellow boxes are sourced from the prior literature.
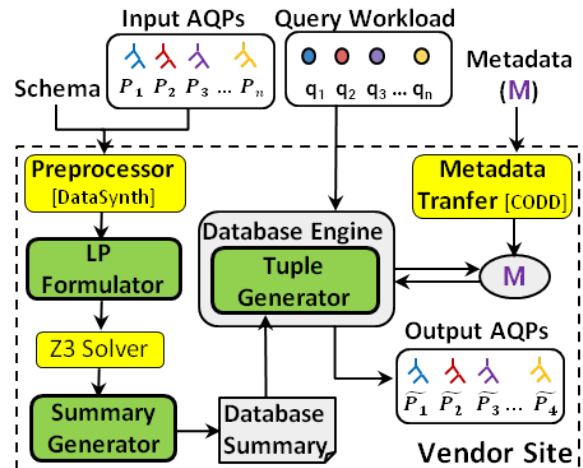

Figure 2: Hydra Architecture

At the client site, Hydra fetches the schema, metadata and the query workload with its corresponding AQPs, and ships this entire information to the vendor. If required, privacy concerns can be addressed by passing the information through an appropriate *anonymization* layer at the client.

When the information is received at the vendor, it initially goes through a **Preprocessor**, sourced from [4]. This component facilitates the *independent* processing of each relation in the subsequent steps, a key requirement for model tractability and regeneration efficiency. The AQPs are then evaluated by the **LP Formulator** and an optimized LP is constructed for each relation, using our new region-partitioning approach. This collection of LPs is passed to the Z3 solver [10], which provides feasible per-relation solutions as the output. Leveraging these solutions, the **Summary Generator** constructs a summary using our deterministic alignment algorithm. Further, a post-processing step is executed to ensure that referential constraints are not violated across the solutions. This step may incur minor additive errors in satisfying the volumetric constraints, but their impact is expected to be negligible at Big Data scale.
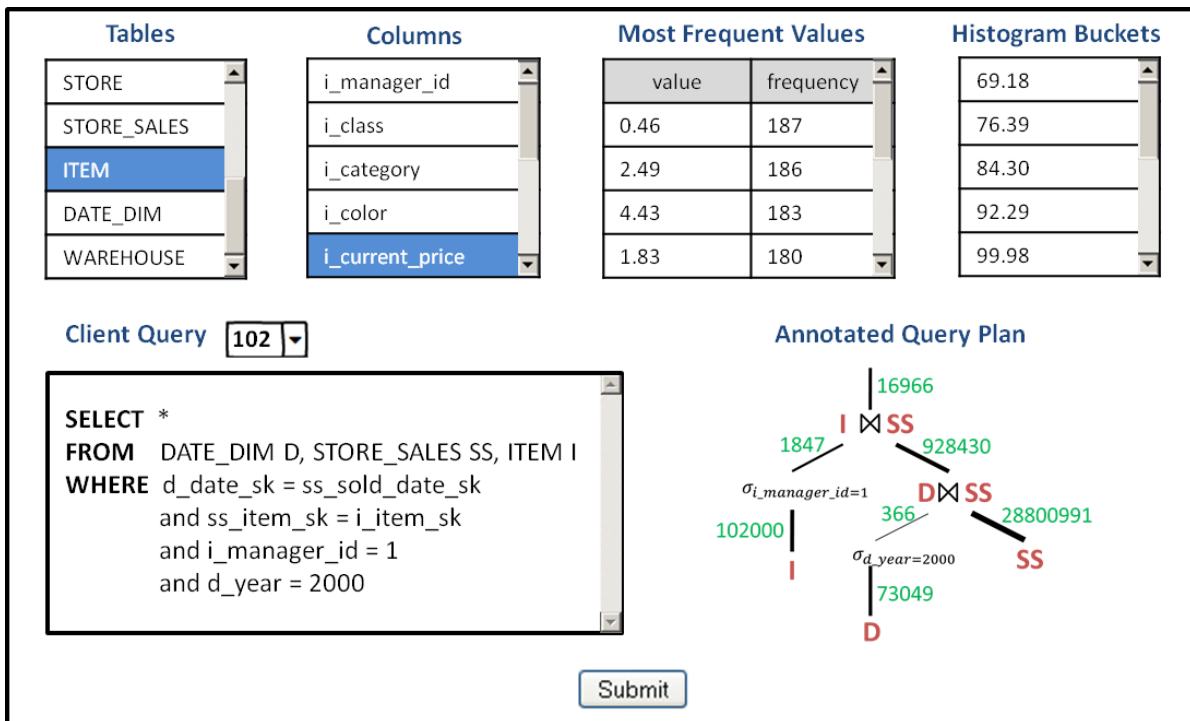
Figure 3: Client Site: Metadata, Queries and Annotated Query Plans

Subsequently, the **Tuple Generator** generates the requisite data *on-demand*, one row at a time, for each relation appearing in the query, using the database summary. As a proof of concept, we have implemented this functionality in the PostgreSQL v9.3 engine [3] by adding a new feature called datagen, which is included as a property for each relation in the database. On enabling this feature for a relation, the traditional scan operator is replaced with the equivalent dynamic regeneration operator.

Finally, the metadata transfer functionality of CODD [5] is used to ensure a common choice of plan at the client and vendor sites.

4. HYDRA DEMONSTRATION

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

4.1 Client Site

At the Client Site, the client supplies the query workload and the corresponding AQPs are obtained by optimizing and executing these queries on the client platform. Currently, the JSON format is supported to parse the execution plans. The next screen in the client interface is shown in Figure 3. In this figure, the top half profiles the metadata statistics. Specifically, the user can choose a specific table column, and the system presents the distribution of the most frequent values and the bucket boundaries of the equi-depth histogram for this column.[1] In the bottom half, the user can pick a query from the input workload (the figure shows a canonical SPJ query on the TPC-DS schema), and the corresponding SQL text is displayed at the bottom left along with the associated AQP at the bottom right. The widths of the edges connecting the operators in the plan are scaled to visually indicate the volume of data flowing in each of these edges. Finally, once the user selects the SUBMIT button, all this information is transferred to the Vendor Site.

---
[1]The metadata visualization is customized for PostgreSQL [3].

4.2 Vendor Site

After receiving the above-mentioned information package from the client, the Vendor Site initiates the data regeneration process. Here, the primary interface during the LP solving stage tabulates the LPs complexity in terms of their number of variables and run times. Subsequently, in the next screen, shown in Figure 4, the final database summary is displayed. The user can select an individual relation, and the system shows its summary in the top middle panel. The difference in the schema of a *relation summary* and that of the corresponding relation is that the pk column in the relation is replaced with a #TUPLES column – this column captures the number of tuples that share the vector of data values present in the remaining columns. For instance, the first row in the item relation summary in Figure 4 indicates that there are 917 rows with values <40, pop, Music, ...>. The pk columns are subsequently generated as auto-numbers. Note that this approach does not affect the referential constraints or the AQP constraints, as the foreign-keys have already been assigned compatible values.

Secondly, the top right panel shows the runtime configuration settings where the user can choose to either dynamically generate or optionally materialize the selected relation. Also, for dynamic generation, the desired velocity, measured in rows per second, can be set using the slider bar. The chosen relation's row count for the original and synthetic database are shown below the bar. To assess the overall quality of the regenerated data, the bottom left graph plots the percentage of volumetric constraints that are satisfied within a given relative error. Finally, the user can also drill down to a query-specific AQP comparison by selecting a query from the drop down menu. In this mode, the corresponding SQL text and AQP are shown in the bottom middle and right panels, respectively. The edges in the AQP are annotated with the original cardinality in green color, and the relative errors (typically minor) incurred as a result of the regeneration are shown in red color.
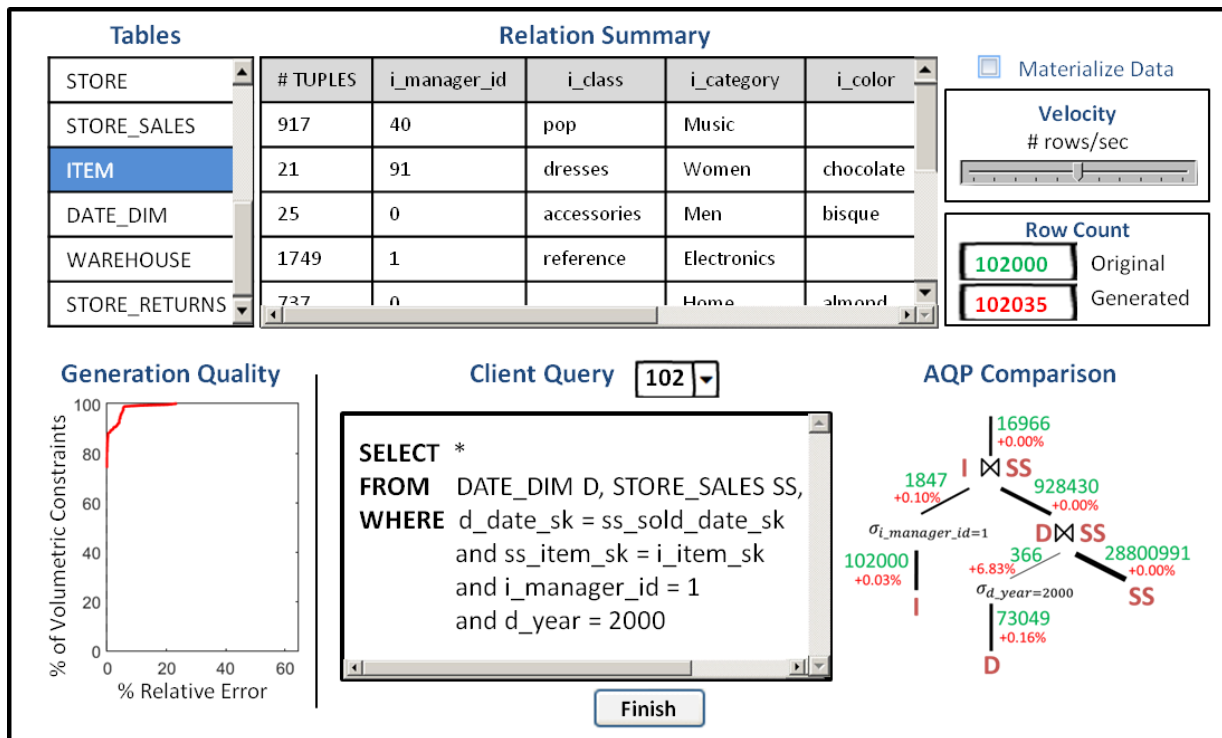
Figure 4: Vendor Site: Database Summary, Runtime Configuration Settings, Generation Quality and AQP Comparison

4.3 Dynamic Regeneration

In this segment of the demo, we explicitly demonstrate that the regenerated database has absolutely *no data* stored in the physical tables – i.e. the "dataless" approach. Instead, using our tuple generator, data is generated and supplied on-demand during query execution. As an example of the final outcome, a few sample rows for the initial columns of the ITEM table (highlighted in Figure 4) are enumerated in Table 1.

Table 1: Sample Tuples

item_sk	i_manager_id	i_class	i_category	...
0	40	pop	Music	...
917	91	dresses	Women	...
938	0	accessories	Men	...
963	1	reference	Electronics	...

4.4 Scenario Construction

Finally, Hydra also facilitates the vendor to pro-actively simulate anticipated client environments, by constructing synthetic AQPs through injecting cardinality annotations into the original client AQPs. For such "what-if" scenarios, Hydra creates the regeneration summary after verifying the feasibility of the synthetic assignments. This feature is particularly useful for testing the ability of the vendor's engine to robustly handle boundary condition scenarios and stressed Big Data environments. In the demo, we will model an extrapolated *exabyte scenario* to showcase this feature, focusing on the efficient summary creation and the on-demand data generation.

5. REFERENCES

[1] Big Data. en.wikipedia.org/wiki/Big_data
[2] Hydra Database Regenerator. dsl.cds.iisc.ac.in/projects/HYDRA
[3] PostgreSQL. postgresql.org/docs/9.3
[4] A. Arasu, R. Kaushik, and J. Li. Data Generation using Declarative Constraints. In *Proc. of ACM SIGMOD Conf.*, 2011, pgs. 685-696.
[5] S. Ashoke and J. R. Haritsa. CODD: A Dataless Approach to Big Data Testing. *PVLDB*, 8(12):2008-2011, 2015.
[6] C. Binnig, D. Kossmann, E. Lo, and M. T. Özsu. QAGen: Generating Query-Aware Test Databases. In *Proc. of ACM SIGMOD Conf.*, 2007, pgs. 341-352.
[7] N. Bruno and S. Chaudhuri. Flexible Database Generators. In *Proc. of 31st VLDB Conf.*, 2005, pgs. 1097-1107.
[8] J. Gray, P. Sundaresan, S. Englert, K. Baclawski, and P. J. Weinberger. Quickly Generating Billion-Record Synthetic Databases. In *Proc. of ACM SIGMOD Conf.*, 1994, pgs. 243-252.
[9] E. Lo, N. Cheng, W. W. Lin, W.-K. Hon, and B. Choi. MyBenchmark: generating databases for query workloads. *The VLDB Journal*, 23(6):895-913, 2014.
[10] L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proc. of TACAS Conf.*, 2008, pgs. 337-340.
[11] T. Rabl, M. Danisch, M. Frank, S. Schindler and H. Jacobsen. Just can't get enough - Synthesizing Big Data. In *Proc. of ACM SIGMOD Conf.*, 2015, pgs. 1457-1462.
[12] A. Sanghi, R. Sood, J. R. Haritsa, and S. Tirthapura. Scalable and Dynamic Regeneration of Big Data Volumes. In *Proc. of 21st EDBT Conf.*, 2018, pgs. 301-312.
[13] J. W. Zhang and Y. C. Tay. Dscaler: Synthetically Scaling A Given Relational Database. *PVLDB*, 9(14):1671-1682, 2016.