High Variety Cloud Databases

Shrainik Jain, Dominik Moritz, Bill Howe
Computer Science and Engineering Department, University of Washington, Seattle, WA, USA
{shrainik, domoritz, billhowe}@cs.washington.edu

***Abstract*—Big Data is colloquially described in terms of the three Vs: Volume, Velocity, and Variety. Volume and velocity receive a disproportionate amount of research attention, however, variety is frequently cited by practitioners as the Big Data problem that "keeps them up at night" — the problem that resists direct attacks in terms of new algorithms, systems, and approaches. We find that the cloud-based data management platform attracts higher variety workloads, therefore motivating a new classes of High Variety Database Management Systems (HVDBMS). This work provides an operational model of variety emphasizing the complexity of *user intent* as well as the complexity of the data itself. The proposed model captures intuitive notions of variety that are distinct from, and broader than, conventional data integration challenges, establishes criteria for a "High Variety benchmark" that can be used to evaluate competing systems, and motivates new research directions in the design of HVDBMS.**

I. Introduction

In 2001, Doug Laney characterized Big Data in terms of challenges associated with volume, velocity, and variety [24]. Since that time, challenges associated with volume and velocity have received disproportionate attention from the research community, in part due to the ease with which these challenges can be translated into system requirements and quantitative experiments: just measure the runtime, parallel efficiency, and fault tolerance in response to increasing the number of bytes of input (volume), the number of bytes of input available per unit time (velocity), or both. The translation of challenges related to *variety* into a coherent systems research agenda, however, has not been as straightforward. Tasks associated with variety are only discussed in colloquial terms; high variety seems to require "munging," "janitor work" [25] and "jujitsu" [29]. A common theme is that high variety implies some combination of diverse users, diverse datasets, or diverse tasks, but a more detailed treatment is not available.

This immaturity in our understanding is concerning: there is evidence that variety, rather than volume or velocity, dominates costs in practice. In an interview study in 2014, Kandel *et al.* showed that data integration is a critical obstacle in the enterprise and that existing tools do not scale to accommodate the diversity of data [19]. In earlier work, Kandel *et al.* argued that data variety causes analysts to spend a significant amount of time manipulating data and assessing its quality [18]. Stonebraker *et al.* identified variety as the bottleneck to practical data science activities and introduced a new data curation system [33] to partially address the problem.

Cloud-based multi-tenant data management and sharing services such as Fusion Tables [13], DataHub [4], or SQLShare [17] provide a natural platform for high-variety data: Users can upload data with varying degrees of structure and make use of general services for discovery, querying, sharing, version control, and more. The utility of these services tends to increase with the number of users, datasets, and applications the system attracts. Query recommendation [22], dataset discovery [27], structure inference [18] and visualization recommendation [36], [28], [21] services are more effective when the models that power them can be trained on a large and diverse corpus. Scheduling and resource allocation algorithms (e.g., [23]) have more flexibiity when there is a large and diverse queue of queries and jobs from which to draw. Sharing and collaboration features [17], [4] become more useful through network effects when there is a critical mass of users. These systems *rely* on diversity in the users, datasets, and workloads rather than just *tolerate* it; they are perhaps nascent examples of what Taleb calls anti-fragile systems [34].

Describing a model that captures the benefits these systems offer in terms of productivity is a goal of this paper. Variety is often described in terms of its "non-relationalness," ignoring the complexity we see even within rows-and-columns data. But many cloud-based data sharing systems are "just a bunch of tables" (JBOT) environments[1] — relational databases with no pre-engineered schema, weak typing (e.g., numeric data represented as strings), non-semantic or missing column names, sentinel string values such as "N/A" to indicate missing data, duplicate rows, and sometimes even significant structural errors such as rows shifted to the right by one or more columns. There are no explicit relationships between tables, and logical entities are scattered across many semi-redundant tables. We refer to this data as "weakly structured" to distinguish it from "unstructured" or "semi-structured." Weakly structured datasets are quite common, emerging from ad hoc analysis sessions, automatic extractions from semi-structured sources, or multi-tenant use of a shared database by non-experts. These scenarios represent a very common form of high variety, but one that is largely ignored by systems that focus on graphs, text, or key-value data.

A number of features have been proposed to assist with weakly structured data. These features fall roughly into one of four categories:

- *Structure inference* features address variety by partially automating the wrapping, scraping, and parsing steps required to bring data into a managed environment with a common data model. Wrangler [18], Open Refine [1], and Tamr [33] are recent examples of this approach, and the large body of work on wrapper induction (e.g., [7]) is relevant as well.
- *Schema-on-read* features (including aspects of MapReduce, most NoSQL systems, and various document- and object-centric databases) purport to address variety by eliminating

[1]In reference to JBOD — just a bunch of disks

the need to pre-define a schema before ingesting data, instead requiring programmers to apply schema information implicitly in the code that manipulates the data.

- *Pay-as-you-go* features help address variety by allowing users to postpone decisions about structure, relationships, or semantics until after the data is loaded. Franklin, Halevy, and Maier adopted this view as part of their dataspace concept [11]. These systems may use relations, graphs, attribute-values, or objects as the underlying data model. Google Fusion Tables [13], Freebase [5], SQLShare are examples of systems that encourage users to define relationships, views, types, constraints, and metadata as a side effect of use rather than pre-defined prior to ingest.
- *Data integration* features apply schema-matching and schema-mapping techniques to assist an analyst in identifying relationships between schemas and generating code to exchange data. The literature in data integration is vast, but OpenII [32] is a representative open source library.
- *Recommendation* features use interaction logs and heuristics to simplify query authoring [22] or aplication assembly [36], [28], [21].

What these features and systems have in common is that they attempt to balance a trade-off among setup costs (e.g., designing a schema and ingesting data), software development costs (writing scripts and queries to answer a question), manual effort (e.g., issuing commands or manipulating a GUI to move data from one system to another or execute a query), and execution costs (computational resources required to run the scripts and compute the answer). These costs motivate a quantitative model of variety; defining and evaluating such a model is the subject of this paper.

Our aim in this paper is to describe a model for *high-variety data management systems* (HVDMS) by analyzing data and workloads from three different systems representing a spectrum of intuitive notions of variety: a scientific database, a multi-tenant data sharing service called SQLShare, and a conventional enterprise RDBMS schema as a low-variety control. Our intuition about high variety in a relational setting comes from studying multi year user written queries on SQLShare [17]. We believe that the SQLShare workload is one of the only publicly available high variety workload. Our central observation is that high variety is as much a function of *user intent* as the data itself. Consider the data on your personal laptop: Certainly many different formats, usage scenarios, and semantics are represented. But few if any of your tasks require accessing, let alone integrating, all of these datasets to answer a particular question. If one never needs to write a program to manipulate all the data, its ostensible variety is irrelevant, and new data management features are not necessarily warranted.

Building on this intuition, we assume users are data scientists who are provided a *corpus* of data and one or more *questions* to answer. Under these assumptions, we can compare solutions in terms of the effort required to answer the given questions over the given corpus. Effort can be modeled in various ways: the sum of development time and execution time, lines of code, cyclomatic complexity, etc. We will make this model precise in §III and show how to use the model in practice in §IV.

One might argue that this model is too general to be useful. After all, measuring programmer productivity and designing tools and languages to maximize these measures has been the central goal of the software engineering community (and arguably all of computer science) for decades. But what has changed in the last five years is that the focus has shifted from software engineering (with a goal of producing a long-term solution) to one of ad-hoc data analysis (without necessarily assuming the solution will be reused in the future). Code produced to answer a Big Data question will not necessarily never be used again; it is retained for provenance, documentation, and reproducibility purposes only, if at all. As data volumes and complexity increase, data analysis tasks that were perhaps once simple manual operations (in, say, Microsoft Excel) or short scripts (in, say, Stata [3]) become significant engineering challenges involving scripts to scrape data from the web, queries to extract data from databases, MapReduce jobs to process unstructured data, and more.

Minimizing the effort required to answer a data science question, and ensuring that this effort scales sub-linearly as new data and new questions are presented to the system, is the fundamental design goal for high-variety data management systems. In this paper, we make the following contributions:

- We present an operational model of variety and show that it captures common intuitions.
- We instantiate this model to compare three relational workloads: an intuitively low-variety enterprise RDBMS scenario, an intuitively high-variety scientific database, and an intuitively high-variety web-based data sharing system with no central schema.
- We propose and evaluate a set of new metrics for quantifying the complexity of a relational workload and schema.
- We use these results to inform a set of requirements for high-variety data management systems, and consider how they apply to some existing cloud-based JBOT services.

We intend for the model we propose to be used to quantify variety problems and compare competing solutions. To make the model concrete, we show how to implement it to compare relational database applications. In a relational context, we consider various ways of measuring the complexity of the data and query workload, show how these metrics relate to effort, and use the results to derive a set of requirements for systems purporting to manage variety.

II. Related Work

Previous definitions of Variety fall under one of the following patterns:

- High variety means anything non-relational [8], [37].
- High variety means relational data integration [1], [18], [19].
- High variety means anything poorly supported by existing tools [26].

Laney defined high variety as indicating a "variety of incompatible formats, non-aligned data structures and inconsistent semantics"[24]; this description still seems apt today. Daveport *et al.* described variety as unstructured data from multiple sources[8]. Zikopoulos *et al.* also emphasize varied sources — logfiles, sensors, email, documents, videos that does not lend itself to processing with conventional tools [37].

Madden describes Big Data as being “too big, too fast or too hard for existing tools to process”[26], which correctly evokes the effort involved in working with the data. However, the *tools* are the actors in this definition rather than the user, ignoring the programmer’s effort as a component of the problem. For example, weakly structured relational data might be considered trivial by this definition since an RDBMS is a viable solution. Further, this definition is no help in comparing and contrasting particular tools — once a tool can “process” the data, the data ceases to be “big” by definition! Structure extraction tools cast the variety problem as one of parsing complex formats to produce (weakly) structured data for further processing. OpenRefine[1] and Wrangler[18] are examples of this approach. These tools offer no support for working with multiple datasets or managing complexity once the parsing step has been completed, which has been shown to be a dominant cost [19]. Classical data integration techniques are related to high variety. The central goal of these approaches is to derive a mediated schema from two or more source schemas, allowing all source data to be queried uniformly[9]. Tools and algorithms in this area induce relationships between database elements (tables, columns, rows) and use these relationships to rewrite queries or restructure data. Despite a long history of research (and a detour through XML in the early part of this century), these techniques do not seem to be widely used by analysts today, in part because of the assumptions that the input schemas are carefully engineered, information carrying structures on which the algorithms can gain purchase. Dataspaces [11] represented an attempt to capture important aspects of the high variety problem, but focused heavily on enterprise settings and managed environments rather than the ad hoc, one-off analysis that characterizes data science activities we see in practice. In contrast to existing definitions, we define high variety in terms of the effort required to answer a set of questions. Using effort as a guide, we reason about tools, environments, and solutions in terms of how much they *reduce* this effort. *User intent* is therefore a factor in deciding variety. We will make this definition precise in the next section.

III. Modeling Variety

We have argued that existing definitions of variety are too informal to use to evaluate and compare systems, approaches, or applications. We propose a simple operational effort-oriented model intended to be general enough to capture the diversity of natural application scenarios, but that can be instantiated in specific scenarios to provide a quantitative basis of comparison.

We consider four sources of effort in undertaking a data science problem: 1) the human effort to understand the relevant data sources, 2) the human effort to write code to extract, integrate, analyze the data, 3) the human effort to perform any manual tasks required (e.g., opening files, loading data, copying and pasting values), and 4) the computational effort to execute the code. The overall effort (usually interpreted as time) to solve the problem is the sum of these four sources. The effort is dependent on the information content of the metadata (schema, catalog, or other organization of data sources, if one exists) S, the information content of the data described by the metadata D, and the information content of the set of tasks T the users wish to accomplish. The metadata S may be one or more relational schemas, the directory structures and file formats holding a set of files, a list of relevant web resources, some combination of these, or any other information one must understand to access the data. The complexity of these inputs drive the effort required to solve the problem. Different data manipulation environments (SQL + RDBMS, R + files, Tableau + Spreadsheets, etc.) are associated with different effort functions over S, D, and T. It is worth noting that the human effort is subjective and will vary based on programming skills, however, we are trying to capture the minimum possible effort required by any user. For coding tasks, effort is a function of time required to code and number of lines of code.

We propose a Variety Coefficient V with respect to an environment Env as

$$\begin{aligned} V_{Env}(D,S,T) = \Omega_{catalog}(S) + \Omega_{code}(S,T) + \\ \Omega_{manual}(D,S,T) + \Omega_{execution}(D,T) \end{aligned} \tag{1}$$

where $\Omega_{catalog}(S)$ is the effort to comprehend the catalog, $\Omega_{code}(S,T)$ is the effort to write the code, $\Omega_{manual}(D,S,T)$ is the effort to perform any manual work, and $\Omega_{execution}(D,T)$ is the effort to run the code. Together, the sum of these dimensions determines V.

Different programming environments lead to different effort functions. Consider the following simple examples:

- An intern is given a CSV file containing all orders from 2014 and is asked to predict the revenue from large orders next month. The intern opens the data as a spreadsheet, manually copies all rows for large orders to a new sheet, writes a formula to extract the month from the order date, then uses a pivot table to compute the average revenue by month from large orders.[2] Finally, the intern takes the average of the prior months as a prediction for next month. The effort to browse the catalog was *low*; it could be modeled as the number of columns in the dataset. The effort to write “code” was also low — just a couple of formulas. The manual effort required is potentially high, in some cases scaling linearly with the number of rows. The execution effort — modeled as wall-clock time — is essentially zero in this example.
- A statistician is given the same task. She writes a script in R to process the data and build a simple regression model that considers seasonalities. The effort to browse the catalog is the same. The effort to write the code is relatively high. The manual effort is close to zero — just opening R. The execution effort depends on the size of the data, but turns out to be significant in this case.
- The company is acquired. The data sciences team is asked to repeat the analysis after combining the original data with a new database with two orders of magnitude more orders stored in a data warehouse. The task is given to a database administrator, who loads the original spreadsheet into a temporary table and writes a series of SQL queries to compute the result. The effort to browse the catalog is high — the warehouse schema is complex. The effort to write the necessary queries is on par with the efforts to write the original scripts, but the queries now scale to data volumes that would overwhelm the original R script.

[2]Not all of these steps are required, but this path represents a typical non-expert’s approach.

The manual effort includes the time to create a temporary table, parse and clean the spreadsheet, and ingest the data — perhaps an hour. The execution time is low — the database is well-engineered.

We want to use the model to reason about the trade-offs in these scenarios: Is the effort to write the R script more than the effort to process the data manually in Excel? The answer depends on how we instantiate the model with concrete effort functions. For example, the effort function Ω_{code} could be derived in at least three ways:

- We might monitor data scientists to directly measure time per task.
- We might infer sessions from the query log, and use the timestamps to estimate development time [22].
- We might estimate effort by the number of lines of code, the cyclomatic complexity, or other *proxy metrics*.

In this paper, we consider the use of proxy metrics to estimate effort. Just as optimizers estimate execution cost from statistics, we estimate effort costs using analogous statistics.

A. Applying the Model

In this section, we explain this model with the help of four illustrative examples and show how they affect the different components of the Variety Coefficient V. For each example, we give an example of instantiating the model with specific effort estimates, illustrating how the model might be used in practice.

The first example considers a user with a relational database (with a well engineered schema) who wants to know the average revenue by month. In the second example a data analyst has two files with $1K$ rows each in Microsoft Excel files. To complete one task the data analyst has to join the two files, create a new column as the sum of two columns and plot the results. The third example is that of SQLShare[17], an ad-hoc schemaless multi-tenant data sharing platform, with a cloud setup with 95 existing datasets and 5 datasets the data analyst has to upload. Based on 10 datasets (5 existing, 5 new), she has to write queries for 10 distinct questions collected from her managers. The last example assumes the same tasks from the third example but considers an analyst who does not have access to a database and writes scripts instead. In these examples, we assume effort is interpreted as development time. $\omega_{catalog}$, ω_{code}, ω_{manual} & $\omega_{execution}$ are the instantiated values of $\Omega_{catalog}(S)$, $\Omega_{code}(S,T)$, $\Omega_{manual}(D,S,T)$ & $\Omega_{execution}(D,T)$ respectively.

a) Relational DBMS: $\Omega_{catalog}$ is a function of the schema S. In the context of an RDBMS, $\omega_{catalog}$ is the effort required to browse and internalize the schema in order to write a query. We can estimate this effort by considering the size of the schema — a large schema takes longer to understand than a small schema. Ω_{code} is a function of the schema S and the question T and measures the time required to write and debug the query to calculate the average revenue per month. The more complex the query, the longer it takes to write. The complexity of a query might be estimated by its length, the number of operators used in a query, or the number of tables used. We will consider various definitions of query complexity in §IV. The query may be `select month, avg(revenue) from revenues group by month`. The manual effort Ω_{manual}, here only opening the interactive console to access the database, is negligible. Also note that the manual effort is constant; it does not depend on D, S or T. The execution time $\Omega_{execution}$ depends on the data D and the query T and might be estimated as the product of the average execution time and the expected number of executions required during debugging.

b) Excel: In the second example, Excel, $\omega_{catalog}$ is the time to understand the structure of the two files. ω_{code} is the effort of writing the formula to sum two columns, which is negligible. The largest impact on the Variety Coefficient V in this example is ω_{manual} since Excel does not support joins and the data analyst has to manually move rows. ω_{manual} depends on $|D|$ which we assume is 1000. We might empirically determine the amount of time an analyst spends inspecting each row and multiply that by 1000. $\omega_{execution}$ is negligible in this case but Excel is generally considered slow compared to databases or scripts.

c) SQLShare: Similar to the RDBMS example, $\omega_{catalog}$ and ω_{code} are the time needed to internalize the schema and the time required to write a query. In this case, the schema is potentially larger, but the cost is perhaps mitigated by keyword search and schema browsing support provided by SQLShare. Since the data analyst writes 10 queries, ω_{code} is proportional to the number of queries $|T|$. The manual effort for SQLShare ω_{code} is uploading the 5 datasets. $\Omega_{execution}$ is the runtime of each query, times the iterations user takes to get it right.

d) Script: $\omega_{catalog}$ is the effort to understand the files. Ω_{code} again scales with $|T|$ but the effort per task is much higher than writing a query in SQLShare. ω_{manual} is the effort of setting up the environment to write custom scripts, which varies depending on the environment. The runtime, $\Omega_{execution}$, compared to a database system will depend on the how much the database can use indexes and other optimizations that are hard to write in scripts. For scripts multiple iterations are needed to get the code right, which increases the ω_{code} and $\omega_{execution}$. Although the workloads and datasets chosen in each example were different, the model provides a way of reasoning uniformly about the costs of data management. This allows us to reason about overall complexities of each system to complete a set of tasks from a user perspective. We summarize these examples in Table I. These examples show ways in which one could measure or estimate the effort to develop a more precise model of variety.

Example	Setup	$\omega_{catalog}$	ω_{code}	ω_{manual}	$\omega_{execution}$
RDBMS	high	medium	medium	very low	low
Excel	none	high	very low	very high	very high
SQLShare	low	medium	high	very low	medium
Script	none	high	very high	very low	high
Fusion Tables	none	medium	very low	high	high
DataHub	low	low	very high	medium	medium
Tableau Public	none	medium	none	high	medium
Wrangler	none	medium	very high	low	medium

TABLE I: Comparison of variety dimensions for various application scenarios.

Also note that with growing data size D, Ω_{manual} only increases for Excel. More tasks T affect Ω_{code} for all examples except for Excel (which has increased Ω_{manual}) but for the RDBMS and SQLShare the effort for each new task is much lower.

IV. Analyzing Variety in Relational Workloads

In §III, we described an abstract model for measuring high variety, but did not instantiate the functions of the variety dimensions $\Omega_{catalog}$, Ω_{code}, Ω_{manual}, and $\Omega_{execution}$ are. In many contexts, it is appropriate to estimate these effort functions in terms of proxy metrics derived from existing workloads. In this section, we consider several candidate proxy metrics and show what they reveal about relevant workloads. We focus on weakly structured relational workloads to simplify comparison between different applications and to focus on semantic rather than structural variety. In an accompanying work [17], we provide algorithms to calculate these proxy metrics. We start with proxies for $\Omega_{catalog}$ in §IV-A and proxies for Ω_{code} & Ω_{manual} in §IV-B. We summarize these proxy metrics in Table II.

A. Estimating Catalog Complexity

The first few proxy metrics we consider can be used to estimate $\Omega_{catalog}$, which models the effort necessary to understand the schema or catalog of the data.

a) Number of Tables: Perhaps the simplest estimate for schema complexity is the number of tables. A larger number of relations increases the number of possible queries and increases the size of the schema. Moreover, it increases the effort required for a user to discover relevant datasets, interpret their structure, and write an appropriate query. Enterprise schemas can become complex, but are relatively small and exhibit explicit, well-defined relationships. Query clients support limited search and discovery features for tables, revealing an assumption that the typical size of schemas is small, and that all tables can be inspected directly. A high variety data management system must lift table discovery up as a first-class feature, as does Google Fusion Tables and similar JBOT systems.

b) Number of Columns: The number of tables ignores the complexity of the tables themselves; a simple extension is to measure the complexity of the schema as the number of columns. For this proxy it is more interesting to look at the maximum and mean of the number of columns rather than the total number of columns in all tables (the dimensionality of the whole database).

c) Other $\Omega_{catalog}$ Proxies: Other proxies that capture $\Omega_{catalog}$ are the data types used by the databases and other structural data definitions such as foreign keys and triggers. A naive way to capture the overall complexity of the schema is to look at the size of the schema definitions in SQL. This method however, only works for our examples because they are all based on SQLServer and thus share a data definition language (DDL). As before, these numbers may be misleading if the schema has a lot of repetitions.

B. Workload Complexity

In this section, we look at two sets of proxies that are describing the complexity of the query workload. The first set is the *number of tasks* and the second is the *query complexity*. The product of those two factors is the *workload complexity*. The workload complexity defines the effort that contributes to Ω_{code}. However, if some actions required by the workload are not well supported and require manual work, Ω_{manual} is affected as well.

1) Number of Tasks: The number of tasks is the first defining factor of workload complexity. Number of unique tasks is directly proportional to Ω_{code} and $\Omega_{execution}$. While the number of repetitive tasks affect $\Omega_{execution}$ alone.

a) Number of Queries: The most straightforward way to count the number of tasks for the three workloads is to count the number of queries. The total number of distinct queries is proportional to Ω_{code}, as distinct queries suggest more code required to get the task done.

b) Number of Query Plan Templates: Queries with the same Structure: Removing duplicate queries by the query string will still keep very similar queries like in TPC-H where only parameters change. Instead of using a string similarity metric, we use the *query plan template*, a representation of the structure of a query to remove duplicates. In addition to the tables and columns accessed by each operator we keep the parameters for each operator but replace all constants with the same value. A high absolute number of unique queries and a high ($\frac{\text{Unique Queries}}{\text{Total Queries}}$) ratio are measures of high diversity of queries in a workload and contribute to Ω_{code}.

2) Query Complexity: The second factor that contributes to workload complexity and thus Ω_{code} is the complexity of the individual queries. There is no consensus on how to measure query complexity so the following proxy metrics are an attempt to capture different facets of query complexity. We look at complexity from a user and a system perspective.

a) Query String Length: The simplest measure of complexity from both the user and system perspective is the length of the query string. The longer the query the more a user has to write and read. Longer queries usually also have more diverse operations and complexity for the system.

b) Runtime: While the length of a query mainly increases the complexity for a user, the runtime of a query could be considered a suitable indicator of complexity from the systems perspective. The Variety Coefficient model has a dedicated factor $\Omega_{execution}$ since runtime is also an indicator for the first Big Data V, Volume. We argue that runtime can be very misleading when evaluating variety since it mostly scales with the amount of data. Consequently, runtime does not independently hint at variety.

c) Number of Operators: To capture the complexity of a query for the system more accurately, we look at the number of operations in the execution plan. More operations mean more steps of computation which increases the complexity of scheduling of data flow for the system. A better metric to capture this case is to look at the *diversity* of operators and count the number of unique operators per query. A combination of both the number of operations and the number of distinct operations intuitively captures complexity better than either of the two metric.

d) Diversity of Operators: In the previous paragraph, we looked at the number of distinct operators for each query. The next question one might ask is what type of the operators are present in the workload as a whole. This metric contributes to the workload complexity as a whole (and thus Ω_{code}) which is primarily of interest to system developers. This metric also contributes to Ω_{manual} and Ω_{code}. If an operator is not supported, the user will have to either use an alternative approach with

Metric	Variety Dimension	Impact
# of Tables & Columns	$\Omega_{catalog}$	Higher values could hint at higher variety, but we need to factor in the effect of compression.
# of Queries, Distinct Queries & Distinct Templates	Ω_{code}	Lower values of the ratios of $\frac{\textit{\# of Distinct Queries}}{\textit{\# of Queries}}$ & $\frac{\textit{\# of Distinct Templates}}{\textit{\# of Queries}}$ suggests lower variety.
Query Length & Runtime	Ω_{code}, $\Omega_{execution}$	Simple measures, could hint at potential high Ω_{code}
No. of operators & distinct operators	Ω_{code}	Better estimator of query complexity, higher values suggest high Ω_{code}
Types of operators & expressions	Ω_{code} & Ω_{manual}	Higher values imply higher Ω_{code} but lack of a certain operator suggests very high Ω_{manual}
Table & Column Touch	Ω_{code}	Higher values increase variety.
Reuse Potential	Ω_{code} and $\Omega_{execution}$	Suggests how diverse the queries are, less reuse potential is more diversity and Ω_{code} and $\Omega_{execution}$
Table Coverage	$\Omega_{catalog}$ and Ω_{code}	Steep curve in the graph hints at lower variety

TABLE II: Summary of how proxy-metrics contribute to variety dimensions.

different operators (Ω_{code}) or manually do the work outside the system (Ω_{manual}) (*e.g.* Microsoft Excel does not support joins). An operation that is not supported at all increases the effort for Ω_{manual} to infinity. Manually executing a join contributes to Ω_{manual} and is directly proportional to the size of the data. Another reason for looking at the type of operators in a workload is that different operators have different complexities. Hence, they require different amount of effort both from a user perspective (understanding and writing queries) and from a systems perspective. Query optimizers already provide cost measurement for different operators, hence we will not focus finding alternate metrics which look at this from a systems perspective. Instead, we want to motivate that, from a user's perspective, different operators contribute differently to Ω_{code}. Also, different operators have different complexities in terms of effort required from user to write them.

e) Diversity of Expressions: In the previous paragraph we stated that users have more difficulty writing some operators than others. However, the complexity is also affected by the parameters of the operator. Scalar computation is a very common operator and the most common operator in SDSS. Similarly, aggregation is (after scan) the most common operator in SQLServer. The scalar operator and aggregate operator use expressions to further specify the operation or aggregation. In this section, we investigate what *expression operators* are used in scalar expressions and aggregates.

f) Table Touch: Number of Tables Referenced per Query: Another proxy measure of complexity, related to schema complexity, is a metric we call *table touch*. It is defined as how many distinct tables are "touched" by a query. A query that uses more tables puts more cognitive load on a user and adds execution complexity which contributes to Ω_{code}.

g) Column Touch: Number of Columns Referenced per Query: Along the lines of the table touch, *column touch* is the number of columns in a query measured as the number of unique column used by the leaf operators in the execution plan. Similarly to the dimensionality of the database in §IV-A this metric describes the dimensionality of a query. Ω_{code} scales with column touch. Science workloads have a) non-experts and experts writing queries, b) unanticipated queries, and c) implicit joins amongst a lot of tables.

3) Reuse Potential: Reducing Runtime: In §IV-B2 runtime is a measure of complexity of a query that a system needs to handle. The runtime however can be reduced and thus affect $\Omega_{execution}$ less if we consider the workload as a whole rather than each query individually. Roy *et al.* show experiments in which 30% to 80% (depending on the workload) of the execution time can be saved by aggressively caching intermediate results[31]. Query optimization in the presence of cached results and materialized views is beyond the scope of this paper. Nonetheless, we implemented a simple algorithm to calculate reuse of query results that matches subtrees of query execution plans. While iterating over the queries, all subtrees are matched against all subtrees from previous queries. We allow a subtree that we match against to have less selective filters (filters are a subset) and more columns for the same tables (columns is a superset). If we find that we have seen the same subtree before, we add the cost of the subtree as estimated by the SQLServer optimizer to the saved runtime. Consequently, a precomputed intermediate result does not cost us anything when being reused. Although this algorithm does not accurately model the actual execution time, we use it to estimate how diverse queries are. The algorithm can underestimate the potential for reuse since the matching misses cases when a rewriting would be needed. It could overestimate since we assume infinite memory as well as no cost for using a previously computed result.

a) Table Coverage: Proportion of Tables Referenced in the Workload: In this section, we explore the relationship between metrics of the workload and the database. Table coverage is the cumulative number of "seen" tables, ordered chronologically by query. This metric expresses how often a new task requires new data. This metric covers $\Omega_{catalog}$ and Ω_{code} on the two axis of the plot. In a low variety system we expect the curve to be steep in the beginning since the first few queries access all or a majority of the available relations.

C. Example Application Scenarios

a) TPC-H: *TPC-H*[6] is a standardized decision-support benchmark consisting of analytical queries. The database schema is fixed and there are only 22 *query templates* designed to reflect typical retail analytic scenarios. These templates can be populated with varying values for the parameters to generate families of related queries. As a result, there is limited variety in either the structure of the data or the queries and thus low values contributing to $\Omega_{catalog}$ and Ω_{code}. We consider TPCH representative of a typical enterprise workload, and in our analysis it represents a low-variety control.

b) Sloan Digital Sky Survey (SDSS): The SkyServer project of the *Sloan Digital Sky Survey*[20] (SDSS) is a redshift and infrared spectroscopy survey of galaxies, quasars, and stars. It led to the most detailed three-dimensional map of the universe ever created at the time. The survey consists of multiple data releases (10 to date), which represent different projects and different stages of processing refinement. SDSS represents one of the few publicly available query workloads from a live SQL database supporting both ad-hoc hand-authored queries as well as queries generated from a point-and-click GUI. The data collected for each data release are publicly available through the SkyServer interface[2]. The schema of the SDSS database was carefully engineered and includes significant use of UDFs and views. SDSS could be considered low-variety due to its reliance on an engineered schema, but the complex workload and table-valued UDFs are atypical for conventional databases and representing an important aspect of variety, as we will say. Intuitively, SDSS would exhibit low values of $\Omega_{catalog}$ but high values of Ω_{code} and $\Omega_{execution}$ due to query complexity.

c) SQLShare: SQLShare[17] is a database-as-a-service system targeting scientists and engineers. Users can upload datasets, write queries across any datasets in the system, and share the results as views. The goal is to reduce the overhead in using relational databases in ad-hoc analytics scenarios: installation, configuration, schema design, tuning, and data ingestion. By uploading a dataset, users extend the schema of the database. If they choose to share the new dataset, others can use it and combine it with their own datasets. Queries can be submitted through a web interface, allowing collaborative query authoring and avoiding any software installation. The SQLShare interface facilitates and encourages the aggressive use of views. Users frequently create deeply nested hierarchies of views to break down complex problems, clean and share intermediate datasets, and share provenance. SQLShare has been deployed in a number of scientific contexts without any explicit SQL training. The use of SQLShare in ad-hoc analytic contexts suggests that it should exhibit higher variety than enterprise RDBMS applications. Specifically, we anticipate that for all dimensions except Ω_{manual} the proposed proxy metrics will exhibit high values.

V. Case Study: SQLShare

§III and §IV talked about the factors which help quantify variety. We can use these features to inform design decisions about new database management systems. Recall that variety can be modeled as the sum of the four 'omegas'. A system designed to lower some or all of the 'omegas' can help tackle high variety. As a case study we talk about why a database-as-a-service platform like SQLShare is better suited to handle relational high variety than other systems. And what additional features can be added to SQLShare to improve it further. Table III shows a summary of SQLShare features and how they affect each of the Ωs. Further details about SQLShare features can found in our companion paper[17]. We believe that success of SQLShare so far can be attributed to how it reduces $\Omega_{catalog}$ and Ω_{manual} by allowing for relaxed schemas, first class views and collaborative sharing. There are two point to note about how SQLShare deals with Ω_{code}. First, SQLShare supports full SQL: This reduces the complexity of code to users being able to express them in SQL. While it true that its not always possible to express complex tasks in SQL, given the nature of datasets (just a bunch of tables) SQL becomes the goto language. In our experience, scientists with little to no background in SQL picked it up in no time [15], [16]. Second, SQLShare can lower Ω_{code} further by incorporating more features in the like 'query recommendation' and 'query auto completion'. $\Omega_{execution}$ is something SQLShare could arguably do better in the future. One possible way to achieve lower $\Omega_{execution}$ is to switch its backend to faster systems like Myria[14], or a federation of multiple database systems as demonstrated in BigDAWG[10].

TABLE III: Summary of SQLShare features.

Feature	Requirement	Ω lowered
Database as a Cloud Service	Diverse users	$\Omega_{catalog}, \Omega_{manual}, \Omega_{execution}$
Relaxed schemas	Weakly structured data, "One-pass" workloads	$\Omega_{catalog}, \Omega_{manual}$
First-class views	Derived datasets	Ω_{code}
User-controlled permissions	Collaborative sharing	Ω_{manual}
Full SQL	Complex manipulation	$\Omega_{code}, \Omega_{manual}, \Omega_{execution}$

VI. Conclusion and Future Work

In this paper we presented a simple operational model of variety motivated by an empirical analysis of real systems and workloads and based on the idea that variety is more a function of user intent and user effort than of the data itself. We intend this approach to be used to assist evaluation of candidate high variety data management systems (which we believe will be best served on the cloud), and as a call to arms for the research community to focus on variety with the same level of attention as simpler questions of direct performance — questions are no longer really the bottleneck in practice. The abstract model guides reasoning, but cannot be used quantitatively without modeling the effort functions. We considered the information content of relational query workloads, reasoning about the total opportunity for reuse as a measure of complexity. We found that these metrics confirmed intuition about the relative variety of the systems we considered. Table II summarizes section §IV with respect to the model described in section §III.

a) For Database Designers: There has been traditionally a separation of concerns between parts of a system, which affect the different Ω's. Minimizing $\Omega_{catalog}$ has been a focus of the database community, Ω_{code} of software engineers, Ω_{manual} of HCI researchers, and $\Omega_{execution}$ of systems. Making specialized query language support (or its equivalent there of) decreases both Ω_{code} and Ω_{manual}. It is interesting to note how lack of support for a particular operation in a system (while still having the most optimized algorithms for the operations it does support) can increase Ω_{manual} more than it decreases Ω_{code}. The variety model in this paper suggests that a unification of these efforts is required to address variety concerns.

b) For Data Scientists: We anticipate the model being used to challenge vendors to specify their value proposition: if you claim to address variety challenges, you can articulate the effect on the effort for data scientists — do you reduce the coding effort at the expense of manual effort? Do you improve performance but increase the complexity of the schema? Even for non-experts, this model can serve as a reference point. It can be used as a basis for questions to systems and database developers on how their system fares over the different dimensions of the variety coefficient V.

c) Future work: We performed this initial workload analysis as a step towards a quantitative high variety benchmark based in part on the SQLShare[17] dataset. As a first step, we needed a model of variety that would capture and confirm intuition about different application scenarios in order to justify the use of a particular scenario as exhibiting high Variety. Previous efforts to develop benchmarks focus exclusively on performance, *i.e.*, Volume and Velocity [30], [35], [12], but do not address the variety problems cited as the bottleneck in practical contexts. In future work, we will continue to develop the techniques to analyze and quantitatively measure the variety of workloads, allowing a before and after measurement with which to evaluate systems and algorithms. We are also considering caching opportunities for repetitive workloads in complex schema-free situations. It is also worthwhile to look at better ways to measure Ω_{manual}, something this work mostly ignored. We are also working on detailed analysis of the query connect graphs which we believe will turn out to be a useful way to represent the properties of data and workload combined, and may provide an application "signature" that we can use to cluster problems and recommend solutions. In particular, we will explore using query connect graphs to generate random high variety workloads as the basis for benchmarks. Our hope is that this paper initiates a research agenda in high variety systems and benchmarking.

References

[1] OpenRefine (formerly google refine). http://openrefine.org/. Accessed: 2014-10-14.

[2] Sloan digital sky survey SkyServer. http://cas.sdss.org/. Accessed: 2014-02-12.

[3] Stata. http://www.stata.com/.

[4] A. P. Bhardwaj, S. Bhattacherjee, A. Chavan, A. Deshpande, A. J. Elmore, S. Madden, and A. G. Parameswaran. Datahub: Collaborative data science & dataset version management at scale. *CoRR*, abs/1409.0798, 2014.

[5] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[6] T. P. P. Council. TPC-H benchmark specification. http://www.tpc.org/tpch/, 2008.

[7] N. N. Dalvi, R. Kumar, and M. A. Soliman. Automatic wrappers for large scale web extraction. *PVLDB*, 4(4):219–230, 2011.

[8] T. H. Davenport, P. Barth, and R. Bean. How 'big data'is different. *MIT Sloan Management Review*, 54(1), 2013.

[9] A. Doan and A. Y. Halevy. Semantic integration research in the database community: A brief survey. *AI magazine*, 26(1):83, 2005.

[10] A. Elmore, J. Duggan, M. Stonebraker, M. Balazinska, U. Cetintemel, V. Gadepally, J. Heer, B. Howe, J. Kepner, T. Kraska, et al. A demonstration of the bigdawg polystore system. *Proceedings of the VLDB Endowment*, 8(12):1908–1911, 2015.

[11] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *ACM Sigmod Record*, 34(4):27–33, 2005.

[12] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 international conference on Management of data*, pages 1197–1208. ACM, 2013.

[13] H. Gonzalez, A. Y. Halevy, C. S. Jensen, A. Langen, J. Madhavan, R. Shapley, W. Shen, and J. Goldberg-Kidon. Google fusion tables: web-centered data management and collaboration. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1061–1066. ACM, 2010.

[14] D. Halperin, V. Teixeira de Almeida, L. L. Choo, S. Chu, P. Koutris, D. Moritz, J. Ortiz, V. Ruamviboonsuk, J. Wang, A. Whitaker, et al. Demonstration of the myria big data management service. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 881–884. ACM, 2014.

[15] B. Howe, G. Cole, E. Souroush, P. Koutris, A. Key, N. Khoussainova, and L. Battle. Database-as-a-service for long-tail science. In *Scientific and Statistical Database Management*, pages 480–489. Springer, 2011.

[16] B. Howe, F. Ribalet, D. Halperin, S. Chitnis, and E. V. Armbrust. Sqlshare: Scientific workflow via relational view sharing. *Computing in Science & Engineering, Special Issue on Science Data Management*, 15(2), 2013.

[17] S. Jain, D. Moritz, B. Howe, D. Halperin, and E. Lazowska. Sqlshare: Results from a multi-year sql-as-a-service experiment. In *Proceedings of the 2016 ACM SIGMOD international conference on Management of data*, 2016.

[18] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3363–3372. ACM, 2011.

[19] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Enterprise data analysis and visualization: An interview study. In *IEEE Visual Analytics Science & Technology (VAST)*, 2012.

[20] S. M. Kent. Sloan digital sky survey. In *Science with Astronomical Near-Infrared Sky Surveys*, pages 27–30. Springer, 1994.

[21] A. Key, B. Howe, D. Perry, and C. Aragon. Vizdeck: self-organizing dashboards for visual analytics. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 681–684. ACM, 2012.

[22] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for sql. *Proceedings of the VLDB Endowment*, 4(1):22–33, 2010.

[23] Y. Kwon, M. Balazinska, B. Howe, and J. Rolia. Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 25–36. ACM, 2012.

[24] D. Laney. 3d data management: Controlling data volume, velocity and variety. *META Group Research Note*, 6, 2001.

[25] S. Lohr. For big-data scientists, 'janitor work' is key hurdle to insights. *New York Times*, August 17 2014.

[26] S. Madden. From databases to big data. *IEEE Internet Computing*, 16(3):0004–6, 2012.

[27] K. Morton, M. Balazinska, D. Grossman, and J. Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *Proc. VLDB Endow.*, 7(6):453–456, Feb. 2014.

[28] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina. Seedb: Visualizing database queries efficiently. *Proc. VLDB Endow.*, 7(4):325–328, Dec. 2013.

[29] D. Patil. *Data Jujitsu: The Art of Turning Data Into Product*. O'Reilly Media, 2012.

[30] S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi. Ycsb++: benchmarking and performance debugging advanced features in scalable table stores. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 9. ACM, 2011.

[31] P. Roy, K. Ramamritham, S. Seshadri, P. Shenoy, and S. Sudarshan. Don't trash your intermediate results, cache'em. *arXiv preprint cs/0003005*, 2000.

[32] L. Seligman, P. Mork, A. Y. Halevy, K. Smith, M. J. Carey, K. Chen, C. Wolf, J. Madhavan, A. Kannan, and D. Burdick. Openii: an open source information integration toolkit. In A. K. Elmagarmid and D. Agrawal, editors, *SIGMOD Conference*, pages 1057–1060. ACM, 2010.

[33] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu. Data curation at scale: The data tamer system. In *CIDR*, 2013.

[34] N. N. Taleb. *Antifragile: Things that gain from disorder*, volume 3. Random House Incorporated, 2012.

[35] E. Walker. Benchmarking amazon ec2 for high-performance scientific computing. *Usenix Login*, 33(5):18–23, 2008.

[36] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *Visualization and Computer Graphics, IEEE Transactions on*, 22(1):649–658, 2016.

[37] P. Zikopoulos, C. Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.