Collaborative Data Analytics with DataHub

Anant Bhardwaj
MIT
anantb@csail.mit.edu

Amol Deshpande
U. Maryland (UMD)
amol@cs.umd.edu

Aaron Elmore
U. Chicago
aelmore@cs.uchicago.edu

David Karger
MIT
karger@mit.edu

Sam Madden
MIT
madden@csail.mit.edu

Aditya Parameswaran
U. Illinois (UIUC)
adityagp@illinois.edu

Harihar Subramanyam
MIT
hsubrama@mit.edu

Eugene Wu
Columbia
ewu@cs.columbia.edu

Rebecca Zhang
MIT
ryzhang@mit.edu

ABSTRACT

While there have been many solutions proposed for storing and analyzing large volumes of data, all of these solutions have limited support for *collaborative data analytics*, especially given the many individuals and teams are simultaneously analyzing, modifying and exchanging datasets, employing a number of heterogeneous tools or languages for data analysis, and writing scripts to clean, preprocess, or query data. We demonstrate DataHub, a unified platform with the ability to load, store, query, collaboratively analyze, interactively visualize, interface with external applications, and share datasets. We will demonstrate the following aspects of the DataHub platform: (a) *flexible data storage, sharing, and native versioning capabilities:* multiple conference attendees can concurrently update the database and browse the different versions and inspect conflicts; (b) *an app ecosystem that hosts apps for various data-processing activities:* conference attendees will be able to effortlessly ingest, query, and visualize data using our existing apps; (c) *thrift-based data serialization permits data analysis in any combination of 20+ languages, with DataHub as the common data store:* conference attendees will be able to analyze datasets in R, Python, and Matlab, while the inputs and the results are still stored in DataHub. In particular, conference attendees will be able to use the *DataHub notebook* — an IPython-based notebook for analyzing data and storing the results of data analysis.

1. INTRODUCTION

Organizations collect and acquire data from various sources, such as financial transactions, social interactions, user activity logs, server logs, sensor data, inventory tracking, and so on. Typically, multiple teams or individuals are interested in extracting insights from these datasets, often via their own home-grown tools, and often collaborate on these datasets, making modifications, such as normalization and cleaning, and then exchanging these datasets back and forth. While there are many point solutions that let individuals or teams ingest and analyze large datasets, collaborative data analysis is still excruciatingly difficult due to the heterogeneity in tools involved, the wide diversity in skill-sets of participating individuals and teams, as well as the difficulty in storing, retrieving, and reasoning about the many versions of the exchanged datasets. Consider the following examples, which represent two extreme points in our spectrum of users and use cases:

- *Expert analysis:* Members of a web advertising team want to extract insights from unstructured ad-click data. To do so, they would have to take the unstructured ad-click data, write a script to extract all the useful information from it, and store it as a separate dataset. This dataset would then be shared across the team. Oftentimes, some team member may be more comfortable with a particular language or tool, e.g., R, Python, Awk, and would like to use this tool to clean, normalize, and summarize the dataset, saving the intermediate results in some way. Other more proficient members may use multiple languages for different purposes, e.g., modeling in R, string extraction in awk, visualization in JavaScript. The typical way to manage dataset versions is to record it in the file name, e.g., "table_v1", "table_nextversion", which can quickly get out of hand when we have hundreds of versions. Overall, there is no easy way for the team to keep track of, study, process, or merge the many different dataset versions that are being created in parallel by many collaborating team members using many different tools.
- *Novice analysis:* The coach and players of a football team want to study, query, and visualize their performance over the last season. To do so, they would need to use a tool like Excel or Tableau, both of which have limited support for querying, cleaning, analysis, or versioning. For instance, if the coach would like to study all the games where the star player was absent, there is no easy way to do that but to manually extract each of the games where the star player was not playing and save it as a separate dataset. Most of these individuals are unlikely to be proficient with data analysis tools, such as SQL or scripting languages, and would benefit from a library of "point-and-click" apps that let users easily load, query, visualize, and share results with other users without much effort.

There are a variety of similar examples of individuals or teams who need to collaboratively analyze data, but are unable to do so because of the lack of (1) flexible dataset sharing and versioning support, (2) "point-and-click" apps that help novice users do collaborative data analysis, (3) support for the plethora of data analysis languages and tools used by the expert users. This includes, for example, (a) geneticists who want to share and collaborate on genome data with other research groups; (b) ecologists who want to publish a curated population study, while incorporating new field studies from teams of grad students in isolated copies first; (c) journalists who want to examine public data related to terrorist strikes in Afghanistan, annotating it with their own findings and sharing with their team.

To address these use cases, and many more similar ones, we propose DataHub, a unified data management and collaboration platform for hosting, sharing, combining and collaboratively analyzing diverse datasets. DataHub has already been used by data scientists in industry, journalists, and social scientists, spanning a wide variety of use-cases and usage patterns.

DataHub has three key components, designed to support the above use data collaboration use cases:

I: Flexible data storage, sharing, and versioning capabilities. DataHub efficiently keeps track of all versions of a dataset, starting from the uncleaned, unstructured versions, to the fully-cleaned structured ones. This way, DataHub enables many individuals or teams to collaboratively analyze datasets, while at the same time allowing them to store and retrieve these datasets at various stages of analysis. Recording, storing, and retrieving versions is central to both the use-cases described above. We described some of the challenges in versioning for structured or semistructured data in our CIDR paper [3]. As part of the demonstration, we will provide a web-based version browsing tool, where conference attendees can examine version graphs (encoding derivation relationships between versions), and semi-automatically merge conflicting versions (with suggestions from the tool).

II: App ecosystem for easy querying, cleaning, and visualization. DataHub has an app library with support for many common data-processing activities, including:

- Distill: Distill is a data-cleaning by example tool; the analyst simply provides a small number of examples of how to transform the unstructured data into a structured, tabular form, and the tool generalizes from these examples to extract structured data from the entire dataset.
- DataQ: DataQ is a query builder tool that allows users to build SQL queries by direct manipulation of the table in a graphical user interface. This kind of interface is suitable for non-technical users such as journalists and social scientists for basic operations such as filters, joins, grouping, and ordering.
- DViz: DViz is a visualization tool, enabling analysts to either specify the visualization that they would like to examine, or be automatically recommended a visualization [11].

While novice analysts stand to gain the most from these tools, expert analysts could also benefit if their home-grown tools are insufficient. The goal here is to ensure that each of these data processing activities takes a handful of actions — drag and drops, mouse clicks or keystrokes. As part of the demonstration, we will allow conference attendees to use each of these apps to process data.

III: Language-agnostic hooks for external data analysis. Expert data analysts often have their preferred languages and libraries to explore and manipulate data, such as Python, R, Scala, and Octave. To support data manipulation and analysis in these languages, DataHub uses Apache Thrift [10] to translate between these languages and datasets in DataHub. Thus, data analysts can continue to use these languages while gaining all the storage, versioning, and retrieval benefits that come with DataHub. In addition, DataHub has a notebook feature (called DataHub Notebook), modeled after the IPython Notebook [9], where data analysts can record all the intermediate commands and query results, tightly integrated with the underlying DataHub storage infrastructure. The DataHub notebook provides an extension point to add support for other languages that can interpret thrift objects. As part of the demonstration, we will allow conference attendees to access DataHub datasets in various data analysis languages, and within the DataHub Notebook.

We present our storage architecture in Section 2, and then describe our proposed demonstration of the key components of DataHub in Section 3. Detailed descriptions of related work are omitted due to lack of space, and can be found in the full technical paper [3]; we mention below some specific papers that can be used as a starting point for other papers.

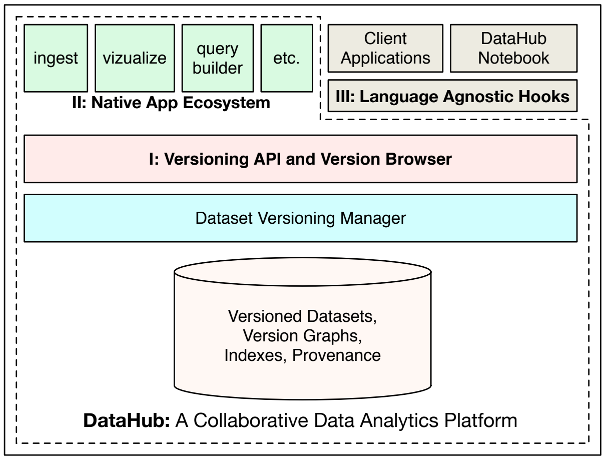

ingest
vizualize
query builder
etc.
II: Native App Ecosystem
Client Applications
DataHub Notebook
III: Language Agnostic Hooks
I: Versioning API and Version Browser
Dataset Versioning Manager
Versioned Datasets, Version Graphs, Indexes, Provenance
DataHub: A Collaborative Data Analytics Platform

Figure 1: DataHub Architecture

Related Work. DataHub is related to work in multiple areas. The overall goal of collaboration is also espoused by fusion tables [5], however, fusion tables does not support versioning or modifications in other external tools. The versioning infrastructure is related to temporal databases (see [3]), however, temporal databases only allow a linear chain of versions (not conducive to collaboration). Versioning is also related to the large body of literature on provenance-enabled databases, e.g., [4], and is related to source-code versioning technologies `git` and `svn`; however, those technologies are not optimized for storing structured or semi-structured data. Some of the apps in the app ecosystem are inspired by related work: DataQ is similar to visual query builders such as Dataplay [2], but opts for a simpler, more direct specification of SQL commands; Distill is similar to data wrangler and flash-fill [7, 6], but gets direct examples from users rather than expecting them to manually perform the operations required for structuring a dataset. DViz is similar to Tableau [8], and our prior work on SeeDB [11].

2. DATAHUB ARCHITECTURE

Figure 1 shows the architecture of DataHub, with the three components described in the introduction. DataHub runs as a server-based "hosted" data management platform that users can connect to to create, ingest, branch, query, and visualize datasets.

Storage Architecture. The main abstraction that DataHub uses is that of a *table* containing *records*. We adopt a simple model where every record in a table has a *key*, and an arbitrary number of typed, named attributes associated with it; we use the word *schema* to loosely refer to the attributes associated with a record. Records are not required to all have the same number/types of attributes. We expect that there will be large groups of records with identical schemas in a table, with a fully cleaned table having the same schema for every record. For completely unstructured data, a single key can be used to refer to an entire file. While there are many other models for data that relax relational "schema-first" requirements, we chose this model because it offers a flexible approach for storing a wide range of data at different levels of structure, allowing both unstructured and fully structured data to co-exist. We do not intend this choice to be limiting or controversial; other "semi-structured" representations, e.g., XML or JSON, would suffice.

The second abstraction is that of a *dataset*. A dataset consists of a set of tables, along with any correlation/connection structure between them (e.g., foreign key constraints).

The versioning information is maintained at the level of datasets in the form of a *version graph*. A version graph is a directed acyclic graph; the nodes correspond to datasets, and the edges capture relationships between versions as well as the *provenance* information recording user-provided and automatically-inferred annotations for derivation relationships between data.

Version API. For any internal and external API to interact with a hosted dataset, it must pass through DataHub's version manager to ensure that reads and writes are directed to the appropriate version. The versioning system is responsible for handling versioning related API requests and maintaining appropriate metadata. Examples of these requests include importing a new dataset (*init*), associating a table with a dataset (*add*), creating a new version of a dataset (*branch*), setting the active version to query against (*checkout*), or merging two divergent versions into a single version(*merge*).

App Ecosystem and APIs. Clients interact with DataHub, and subsequently the versioning system, either through use of a hosted DataHub app, the web-based DataHub application that allows users to explore and query datasets through a web browser, a hosted scripting execution environment, or a remote data access API.

3. DEMONSTRATION COMPONENTS

The goal of our demonstration is to highlight the benefits of having a single consolidated platform for collaborative data analytics. Our demonstration will consist of two end-to-end usage scenarios, one involving novice users, and one involving expert users, both centered on a political contributions dataset [1].

Novice Analysis Scenario Walkthrough. In the first scenario, the demonstration will consist of the following steps, mirroring typical novice user workflows: (1) The uploads a semi-structured document containing political contributions data; this is stored as a new version (Section 3.1 provides details on the web-based version browser); (2) The user cleans and ingest the dataset using the Distill app, which guides the user to semi-automatically generate structured data from the semi-structured dataset (Section 3.2 describes the details of Distill and other DataHub apps); (3) The user "commits" this structured dataset as a new version; (4) The user then decides users other apps for data analysis, including DataQ to query for data of interest and DViz to visually analyze the data.

Expert Analysis Scenario Walkthrough. In the expert usage scenario, the demonstration will consist of five steps: step (1) and (2) are as in the novice scenario. In step (3), the user then exports the dataset into Python using the language-agnostic API (see Section 3.3 for details on this API), modifies it by adding a derived column, and then saves it as a new version. The user repeats this process, by adding or deleting rows, columns, or applying cleaning and normalization steps, saving a number of versions along the way. In step (4) the user realizes that they'd like to go back to a previous version, so they use the version browser to explore and compare versions. Finally, in step (5) the user then decides to document their analysis in the DataHub Notebook (See Section 3.3 for details on the DataHub Notebook.) in order to share it with their team.

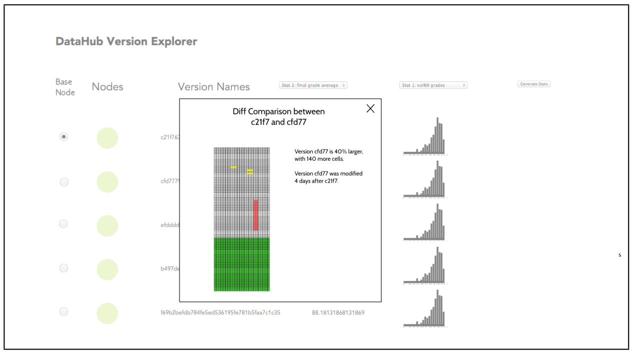
In the remaining sections we provide details of the demo components.

3.1 Version Comparisons and Merging

DataHub allows datasets to be forked and branched, enabling different collaborators to work on their own versions of a dataset and later merge with other versions as needed. Users can, for example, add new records, delete records, apply transformations, add derived columns, delete redundant or useless columns, and so on, all in their own private version of the data, without having to create a complete copy or lose the correspondence with the original dataset.

With the ability to create multiple versions of a dataset, each with unrestricted modifications, complex, divergent datasets may result. As a result, users will often need to visualize and understand the differences between versions. For example, imagine a constantly evolving *master* data set that a user branches from at one point in time. A common use case will be to compare the branch to the master data, to either see what changed in the branch or what new data in the master could be incorporated. Another use case arises when merging two versions to create a single child version, where at some point the two versions shared a common ancestor and can potentially have conflicting changes to the dataset. If there are no conflicts, merging can be done automatically, otherwise DataHub will need to walk the user through the differences.

As part this project we are building a version browser to browse and examine versions, as well as a version graph (displaying how versions have evolved) for both purposes: general differencing and analysis of how versions have evolved, and for merging two versions. Once, say, two versions are selected, the version browser will show a compact representation of changes between the two; coloring will indicate the percentage of records and attributes that have been added, removed, or deleted. In addition to comparing versions, this version browser will allow users to select visualization summaries for certain attributes, such as histograms or pie charts. Starting from this point, the version browser will also allow users to compare and merge versions, using input from the user as needed. Figure 2 shows a screenshot of this demo component. The version graph is on the left, with data histograms on the right. The popup in the front shows additions and modifications between the two versions.



Diff Comparison between c21f7 and cfd77

Version cfd77 is 40% larger, with 140 more cells.

Version cfd77 was modified 4 days after c21f7.

f69b2befdb784fe5ed536195fe781b5faa7c1c35 88.18131868131869

Figure 2: A screenshot of the merging demo

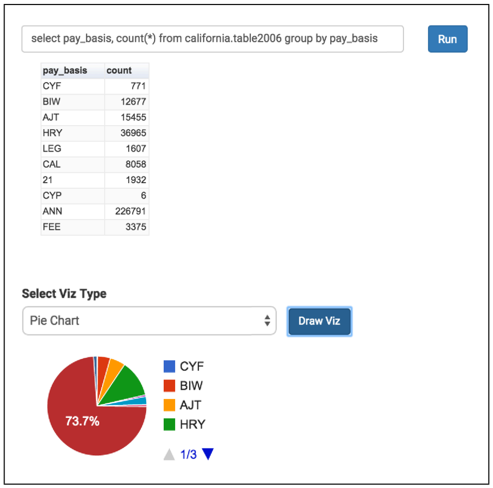
3.2 App Ecosystem

The DataHub app ecosystem hosts apps for various "point-and-click" data-processing activities such as ingestion, curation, querying, and visualization, tailored towards novice (or non-programmer) users. DataHub users can find and use apps from the DataHub App Center for processing their data as it fits their need.

Current apps include:

1. Distill: a general-purpose, example-based data cleaning or extraction tool for converting semi-structured data to a structured table. Here, the user is asked for a few example input-output pairs, and the app generalizes from these pairs to extract from the entire dataset. If the app encounters any difficulties, it will then ask the user for additional targeted examples.

2. DViz (Figure 3): a drag-and-drop interface for creating visualizations. In the screenshot, we show how users can take the result of a query and generate a pie chart from it.

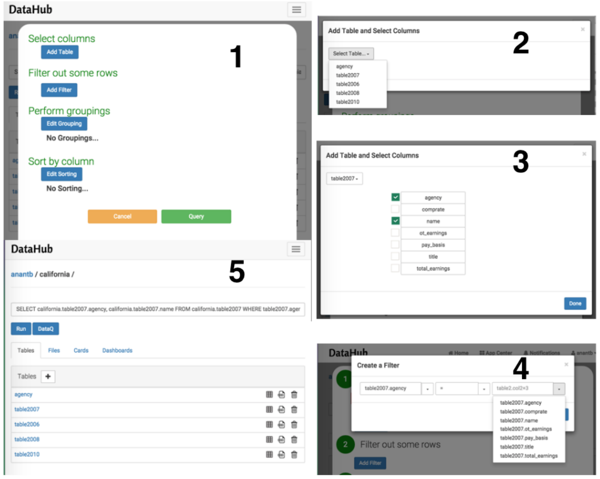


```
select pay_basis, count(*) from california.table2006 group by pay_basis
```

pay_basis	count
CYF	771
BIW	12677
AJT	15455
HRY	36965
LEG	1607
CAL	8058
21	1932
CYP	6
ANN	226791
FEE	3375

Select Viz Type: Pie Chart

Figure 3: A screenshot of the visualization app (DViz)



```
SELECT california.table2007.agency, california.table2007.name FROM california.table2007 WHERE table2007.ager
```

Figure 4: A screenshot of query builder app (DataQ)

3. DataQ (Figure 4): a direct manipulation query interface for non-programmers to pose SQL queries. In the screenshot, the user first adds tables and selects columns, adds filters, then chooses not to perform any aggregation or sorting, and then saves the result as a new table.

We are also in the process of adding apps to enable data integration and to perform simple machine learning tasks such as clustering and classification. A new DataHub app can be written and published to the DataHub App Center using our SDK via thrift-based APIs (see Section 3.3).

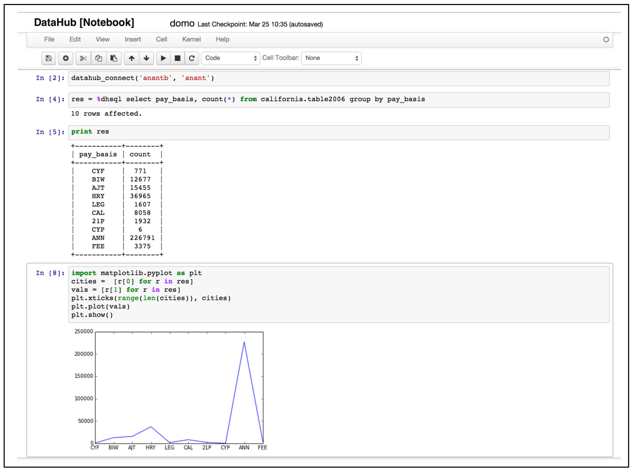
3.3 Language Agnostic Hooks

We now DataHub's programming interfaces and tools for expert users. We first describe the Thrift-based API, followed by the DataHub Notebook.

Remote API. DataHub APIs are described in a Thrift [10] interface description file. Thrift is a framework for scalable cross-language services development. Using Thrift, we are able to generate language bindings for DataHub in a number of different programming languages, including Go, C++, Java, Python, Ruby, and JavaScript. For instance, the Python binding can be generated as follows:

```
thrift --gen py datahub.thrift
```

Once the RPC stubs are generated, a python program can call DataHub APIs. A simple code snippet is shown below.



DataHub [Notebook] domo Last Checkpoint: Mar 25 10:35 (autosaved)

```
In [2]: datahub_connect('anantb', 'anant')

In [4]: res = %dhsql select pay_basis, count(*) from california.table2006 group by pay_basis
        10 rows affected.

In [5]: print res
        +-----------+--------+
        | pay_basis | count  |
        +-----------+--------+
        |    CYF    |  771   |
        |    BIW    | 12677  |
        |    AJT    | 15455  |
        |    HRY    | 36965  |
        |    LEG    |  1607  |
        |    CAL    |  8058  |
        |    21P    |  1932  |
        |    CYP    |   6    |
        |    ANN    | 226791 |
        |    FEE    |  3375  |
        +-----------+--------+

In [8]: import matplotlib.pyplot as plt
        cities =  [r[0] for r in res]
        vals = [r[1] for r in res]
        plt.xticks(range(len(cities)), cities)
        plt.plot(vals)
        plt.show()
```

Figure 5: A screenshot of DataHub Notebook

```
transport = THttpClient.THttpClient('https://datahub.csail.mit.edu/service')
transport = TTransport.TBufferedTransport(transport)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = DataHub.Client(protocol)
print "Version: %s" %(client.get_version())

# open connection
con_params = ConnectionParams(user='anantb', password='*****')
con = client.open_connection(con_params=con_params)

# execute a query
res  = client.execute_sql(
con=con, query='select * from anantb.test.demo', query_params=None)

# print field names
print "\t".join(res.field_names)

# print tuples
for t in res.tuples:
  print "\t".join(t.cells)
```

DataHub Notebook. DataHub notebook (Figure 5) is an extension to IPython [9] to enable interactive analysis using many scripting languages, with the ability to record and store the code along with the analysis results, ideally suited for reproducible data science. The browser-based notebook is integrated with the underlying storage engine of DataHub with support for SQL. DataHub notebook supports custom UDFs, and custom packages written in Python, Javascript, Octave, R, and Scala. The notebook also provides an extension point to add support for other languages which can interpret thrift objects.

4. REFERENCES

[1] Fec presidential campaign finance (http://fec.gov/disclosurep/pnational.do). [Online; accessed 3-March-2014].

[2] A. Abouzied, J. M. Hellerstein, and A. Silberschatz. Dataplay: interactive tweaking and example-driven correction of graphical database queries. In *UIST*, pages 207–218, 2012.

[3] 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. In *CIDR*, 2015.

[4] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[5] 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 *SIGMOD*, pages 1061–1066, 2010.

[6] S. Gulwani, W. R. Harris, and R. Singh. Spreadsheet data manipulation using examples. *Commun. ACM*, 55(8):97–105, 2012.

[7] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. Wrangler: interactive visual specification of data transformation scripts. In *CHI*, pages 3363–3372, 2011.

[8] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional databases. *Commun. ACM*, 51(11):75–84, 2008.

[9] http://ipython.org. IPython (retrieved June 1, 2014).

[10] http://thrift.apache.org. Apache Thrift (retrieved June 1, 2014).

[11] M. Vartak, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: automatically generating query visualizations. *PVLDB*, 7(13):1581–1584, 2014.