Managing Google's data lake: an overview of the GOODS system

Alon Halevy[2*], Flip Korn[1], Natalya F. Noy[1], Christopher Olston[1], Neoklis Polyzotis[1], Sudip Roy[1], Steven Euijong Whang[1]

[1]Google Research

[2]Recruit Institute of Technology

alon@recruit.ai,{flip,noy,olston,npolyzotis,sudipr,swhang}@google.com

Abstract

For most large enterprises today, data constitutes their core asset, along with code and infrastructure. For most enterprises, the amount of data that they produce internally has exploded in recent years. At the same time, in many cases, engineers and data scientists do not use centralized data-management systems and end up creating what became known as a data lake*—a collection of datasets that often are not well organized or not organized at all and where one needs to "fish" for useful datasets. In this paper, we describe our experience building and deploying* GOODS*, a system to manage Google's internal data lake.* GOODS *crawls Google's infrastructure and builds a catalog of discovered datasets, including structured files, databases, spreadsheets, and even services that provide access to the data.* GOODS *extracts metadata about datasets in a post-hoc way: engineers continue to generate and organize datasets in the same way that they have before, and* GOODS *provides value without disrupting teams' practices. The technical challenges that we had to address resulted both from the scale and heterogeneity of Google's data lake and from our decision to extract metadata in a post-hoc manner. We believe that many of the lessons that we learned are applicable to building large-scale enterprise-level data-management systems in general.*

1 Introduction

Most large enterprises today witness an explosion in the amount of data that they generate internally for use in ongoing research and development. The reason behind this explosion is simple: by allowing engineers and data scientists to consume and generate data in an unfettered manner, enterprises promote fast development cycles, experimentation, and, ultimately, innovation that drives their competitive edge. As a result, this internally generated data often becomes a prime asset of the company, on par with source code and internal infrastructure.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Work done while at Google Research.

The flip side of this explosion is the creation of a so called *data lake* [1, 2, 6]: a growing volume of internal datasets, with little codified information about their purpose, value, or origin. This scarcity of information is problematic: data becomes siloed within the teams who carry the "tribal knowledge" of the data's origin, which, in turn, results in significant losses in productivity and opportunities, duplication of work, and mishandling of data.

In this paper we describe Google Dataset Search (GOODS), a system that we built and deployed in order to help Google's engineers organize and manage datasets in its data lake. GOODS operates in a *post-hoc* manner: it collects and aggregates metadata about datasets after the datasets were created, accessed, or updated by various pipelines. Put differently, teams and engineers continue to generate and access datasets using the tools of their choice, and GOODS works in the background, in a non-intrusive manner, to gather the metadata about datasets and their usage. GOODS then uses this metadata to power services that enable Google engineers to organize and find their datasets in a more principled manner. Hence, GOODS is very different from Enterprise Data Management (EDM) systems, which act as gateways and require dataset owners and consumers to use specific protocols and APIs for data access.

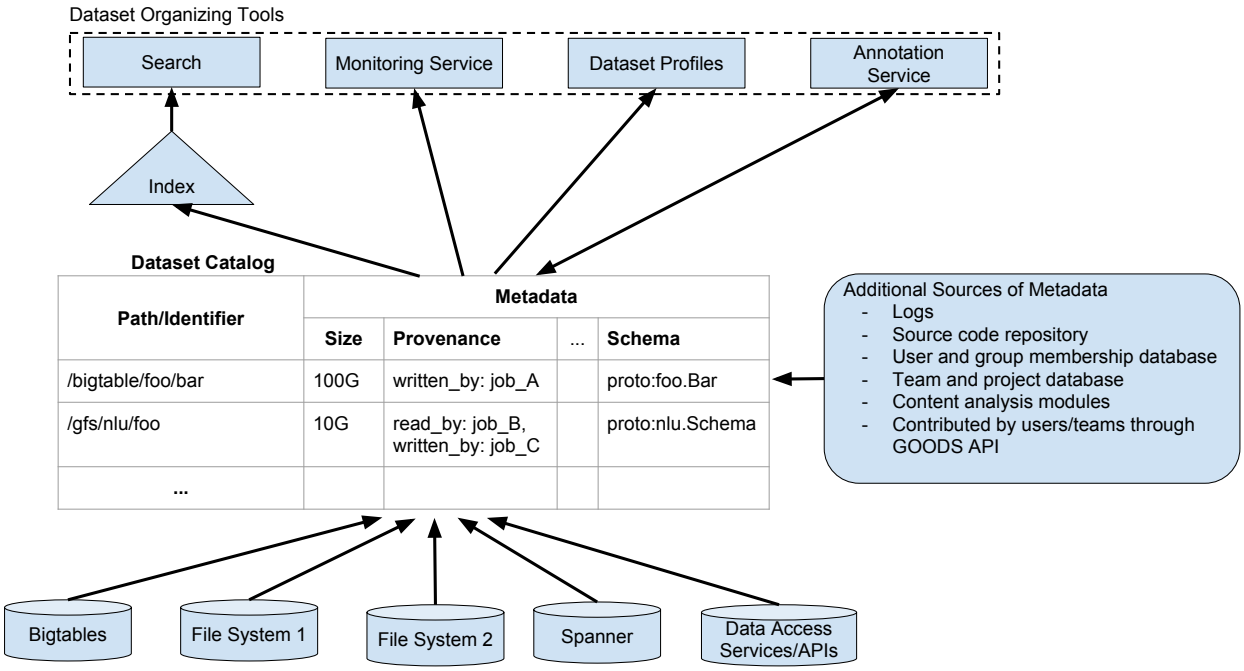

Figure 1: Overview of Google Dataset Search (GOODS). GOODS collects metadata about datasets from various storage systems. It infers metadata and relationships among datasets by processing additional sources such as logs and information about dataset owners and their projects, by analyzing content of the datasets, and by collecting input from the GOODS users. The collected metadata powers user-facing tools, such as search, dataset profiles, monitoring, and a dataset-annotation service.

Figure 1 shows a schematic overview of our system. GOODS continuously crawls different storage systems and the production infrastructure (e.g., logs from running pipelines) to discover which datasets exist and to gather metadata about each one (e.g., owners, time of access, content features, accesses by production pipelines). GOODS aggregates this metadata in a central catalog and correlates the metadata about a specific dataset with

information about other datasets. GOODS then uses this catalog to provide Google engineers with services for dataset management. These services include the following:

- A *search engine* over all the datasets in the company, with facets for narrowing search results, to help engineers and analysts find the most relevant datasets.
- A per-dataset *profile page* that renders the metadata that GOODS has recorded about a specific dataset and can thus help users understand the dataset and its relationships to other datasets in the company. The profile page also integrates with other tools that can further process the dataset, thus helping users act on the data.
- A *monitoring service* that allows teams to monitor features of the datasets that they own, such as size, distribution of values in the contents, or availability. Users can configure this monitoring service to issue alerts if the features change unexpectedly.
- A *annotation service* that allows dataset owners or trusted principals (e.g., data librarians, or a data-stewardship team) to extend a dataset's metadata with domain-specific annotations that can appear in the profile page. As an example, a dataset owner can provide a textual description for the dataset's contents or attach a visualization that can inform other users.

Search is the most frequently used service in GOODS, which demonstrates the importance of dataset discovery within a data lake. However, we have seen good adoption for the remaining services. We were also pleasantly surprised to see that teams used GOODS for scenarios that we did not originally anticipate:

- *Schema auditing* A team used search to identify datasets owned by other teams that conformed to a specific schema that this team owned. The team then audited the resulting datasets to ensure that the schema was used according to its specifications.
- *Data discovery through provenance* The GOODS catalog includes provenance metadata, in the form "dataset Y was read/written by production job X". This information, and in particular the transitive closure of the provenance links, can be useful in understanding the pedigree of a dataset and its downstream dependencies, and thus features prominently in the profile page of each dataset. A team found a different usage for these links and relied on them for dataset discovery. Specifically, a ML team wished to use datasets that were publicized as canonical for certain domains, but they found that these datasets were too "groomed" for ML. To overcome this problem, the team relied on the provenance links to browse upstream and identify the datasets that were used to derive the canonical datasets. These input datasets contained less groomed data and were more suitable for the specific ML task.
- *Content visualization* A technical-infrastructure team used the annotation service to attach a visualization to datasets that represent training data for ML pipelines. This visualization, which illustrates statistics on the features of the training examples, is surfaced on the dataset profile page in order to help users understand the distribution of features and also to spot anomalies that can affect the quality of machine-learned models.

The remainder of the article describes our experience with the development of GOODS. We begin by identifying the key challenges that we had to address in building the data-lake management system at Google—a data lake that contains more than 20 billion datasets. Many of the challenges that we addressed in GOODS were precipitated by the scale and characteristics of the data lake at Google, but we believe that our experience and the lessons that we learned will apply to similar systems in other enterprises. We then introduce the logical structure that we used to organize the data lake's metadata, using relationships among entities (datasets, teams, projects, and so on). We argue that this organization, which is inspired by structured knowledge bases and their

application to Web search, can help answer important questions about the contents of the data lake. We then review briefly some of the system-related and technical issues that we solved in developing GOODS, and finally we present a few directions for future work based on our experience with the current system deployment.

2 Challenges in Managing and Organizing an Enterprise Data Lake

We encountered many challenges as we designed and build GOODS. In this section, we highlight and elaborate on some of the important challenges. A more detailed list of challenges can be found in our previous work [5].

2.1 Organizing the Data Lake

As we discussed, a major goal of a data-lake management system is to gather metadata for the datasets in it. We can view the metadata for each dataset independently. However, we found that is far more powerful to consider how to integrate this metadata in order to uncover relationships among datasets. Identifying and inferring these relationships helped us address some of the scalability challenges that we just described. Furthermore, it helped us organize the datasets in the data lake and thus to provide our users with a better understanding of the data lake's contents.

As an example, consider the following query over the data lake: "Find all datasets that are derived from a dataset owned by project X." This query can help assess the impact of project X or identify teams that need to be notified if there is a plan to wipe the datasets owned by X. Answering this query requires knowledge of the composition of various teams, dataset ownership, and provenance relationships. Another interesting query is "Find all datasets written by a production job whose code uses a specific version of method X", which can help identify datasets that might have been affected by faulty code. Again, answering this query requires knowledge of several types of relationships, including provenance, the versions of binaries in production jobs, and the linkage of source code to binaries.

To support this type of functionality we first have to identify what types of relationships exist and which of them we can determine efficiently. All the challenges that we have identified already still remain. The large scale means that we cannot perform an exhaustive search to identify these relationships. For instance, while it would be useful to know which datasets, or parts of datasets, have identical content, we cannot compare every pair of datasets to each other. Or, because we are building GOODS in a post- hoc manner, we need to understand which infrastructure signals can help uncover these relationships. For instance, to identify provenance relationships between datasets we may join the dataset catalog with logs of different jobs that generate datasets. However, because these resources come from different teams, they often use different naming schemes, different levels of granularity for what they define as a dataset or what they include, and so on.

2.2 Scalability of Metadata Extraction

The first challenge that we had to address is the scalability of of gathering metadata for the datasets in the data lake. Three factors contribute to this challenge: the sheer number of datasets that GOODS manages, the daily churn of the datasets, and the cost of extracting metadata for each individual dataset. Our recent snapshot of Google's data lake contained 20 billion datasets, and this number has likely increased since then. Furthermore, we observed that one billion datasets were added to or removed from the data lake each day, with a significant fraction corresponding to transient short-lived datasets. At the same time, the cost of metadata extraction for individual datasets is high, because this extraction typically requires an expensive operation to the source storage system (e.g., accessing the contents of a file in a distributed filesystem). These factors lead to a prohibitive cost to analyze each and every dataset in the data lake.

One way to address this challenge is to prioritize metadata extraction so that the catalog of the data-lake management system covers the "important" datasets. Coming up with a good metric of dataset importance is

difficult. For example, this metric may depend on the type of the dataset, the context in which the dataset is used (e.g., datasets that power user-facing services may be more important), or relationships to other datasets (e.g., datasets may be more important if they are used to derive other datasets). Similarly, while one may argue that transient datasets are not important given their limited lifespan, we discovered that this is not always the case: In some cases it was necessary to analyze short-lived datasets in order to derive metadata for non-transient datasets. For example, provenance links between non-transient datasets often go through temporary transient datasets.

2.3 Post-hoc Management of Metadata

Early on in the design of our system we decided to adopt a *post-hoc* approach to the process of metadata extraction: the system would gather metadata about datasets by analyzing signals from Google's infrastructure (e.g., by processing logs or crawling storage-system catalogs) after these datasets have been accessed or updated. We can contrast this approach with traditional Enterprise Data Management (EDM) systems, which prescribe specific APIs to access datasets and thus act as gateways between teams and their data. EDM systems can gather very precise metadata because they are in the critical path of data access, but they also require an enterprise-wide opt in. Instead, we designed GOODS to operate from the sidelines assuming that teams were free to choose how they access their data. Our goal was to organize the data lake and to bring value to Google's teams without disrupting their practices. We also believe that this post-hoc approach is in line with the nature of a data lake, which allows engineers and analysts to experiment with data in an unfettered fashion.

The downside of this approach is that it becomes more difficult to extract and reason about dataset metadata. First, we now have to deal with uncertainty in the metadata. As an example, consider the problem of inferring a schema for the contents of a dataset whose storage format does not record this information (e.g., a file containing serialized protocol buffers [7]). The analysis of the contents may yield several candidates for the schema, corresponding to different ways to parse the contents, and in the absence of other information we record all of these candidates in the dataset's metadata. Second, the metadata that we collect is heterogeneous because different types of datasets that appear in the data lake (e.g., files, spreadsheets, relational databases, or instances of key-value stores) have different metadata and may require different tools to extract it. A data-lake management system must be able both to extract and handle metadata across a variety of source storage systems and to record this heterogeneous metadata in a single catalog. However, having a single catalog for diverse metadata is also an opportunity to partially lift this heterogeneity in the data lake: the catalog can define a subset of the gathered metadata that is often common across datasets of different types. This common metadata can provide users and services with a unified view of the datasets in the data lake.

3 The Data Lake Relationship Graph

We view the GOODS catalog not only as a collection of metadata describing the datasets in the Google's data lake but also as a representation of relationships among datasets and other related entities (teams, projects, code, and so on).

This approach is inspired by the concept of *knowledge graphs* [3], which are used by modern enterprises to describe entities in the real world and to allow users to search with complex queries. Nodes in such a knowledge graph represent entities in the world (e.g., Tom Hanks, "Forrest Gump") and edges link these entities to each other (e.g., Tom Hanks played a role in "Forrest Gump"). This graph enables an extensible and flexible representation and supports queries such as "female actors who played lead roles in comedies," which require traversing diverse relationships. We can view the structure of a data lake in a similar way, in particular as we link it to other components on enterprise infrastructure. First, we can have relationships between datasets (e.g., dataset A is a new version of dataset B). Second, we can link datasets to other entities, such as jobs that generate the datasets, projects and users that own the datasets and these jobs, or source code that defines these jobs. As

we list the relationships that we identified between datasets in a data lake, it is important to note that we infer all these relationships automatically, in a post-hoc fashion, relying on a variety of information that we can gather inside the enterprise.

The following is a list of relationships among datasets that we identify as important and that we infer as we collect the metadata in the GOODS catalog.

Dataset containment: Some datasets may contain other datasets. For instance, bigtable column families[4] are first-class entriies in the GOODS catalog, and so are the bigtables themselves. We link the latter to the entries for the column families that they contain. This containment information is usually part of the metadata that we can extract directly from specific storage systems.

Provenance: Datasets are produced and consumed by code. This code may include analysis tools such as dashboarding solutions or SQL-query engines, serving infrastructures that provide access to datasets through APIs, or ETL pipelines that encode dataset transformations. For each dataset, we maintain the provenance of how the dataset is produced, how it is consumed, what datasets this dataset depends on, and what other datasets depend on this dataset. We identify and populate the provenance metadata through an analysis of production logs, which provide information on which jobs read and write each dataset. We then create a transitive closure of this graph connecting datasets and jobs, in order to determine how the datasets themselves are linked to one another. However, the number of data-access events in the logs can be extremely high and so can be the size of the transitive closure. Therefore, we trade off the completeness of the provenance associations for efficiency by processing only a sample of data-access events from the logs and also by materializing only the downstream and upstream relations within a few hops as opposed to computing the true transitive closure.

Logical clusters: We identify datasets that belong to the same *logical cluster*. While our definition of clusters is domain-dependent, in general, we usually group the following collections of datasets into a single logical cluster: datasets that are versions of the same logical dataset and that are being generated on a regular basis; datasets that are replicated across different data centers; or datasets that are sharded into smaller datasets for faster loading. Because engineers tend to use specific conventions in naming their datasets, we can identify these logical clusters efficiently by examining the dataset paths. For example, consider a dataset that is produced daily and let `/dataset/2015-10-10/daily_batch` be the path for one of its instances. We can *abstract* out the day portion of the date to get a generic representation of all datasets produced in a month: `/dataset /2015-10-<day>/daily_batch`, representing all instances from October 2015. By abstracting out the month as well, we can go up the hierarchy to create abstract paths that represent all datasets produced in the same year: `/dataset/2015-<month>-<day>/daily_batch`. By composing hierarchies along different dimensions, we construct a *granularity semi-lattice* structure where each node corresponds to a different *granularity* of viewing the datasets.

Content similarity: Content similarity—both at the level of dataset as a whole and at the level of individual columns—is another graph relationship that we extract. Given the size of Google's data lake, it is prohibitively expensive to perform pairwise comparison of all datasets. Instead, we rely on approximate techniques to determine which datasets are replicas of each other and which have different content. We collect fingerprints that have checksums for the individual fields and locality-sensitive hash (LSH) values for the content. We use these fingerprints to find datasets with content that is similar or identical to the given dataset, or columns from other datasets that are similar or identical to columns in the current dataset.

In addition to the relationships that link datasets in the GOODS catalog to each other, we also rely on the rest of Google infrastructure to enrich this relationship graph. While this part of the system continues to grow, we list here some of the relationships that we currently extract:

- information on owners of the jobs that produce or read datasets;
- information on dataset owners and user groups that determine visibility permissions for datasets;
- links between schema that we infer for the datasets and its definition in the source code repository.

Using this (conceptual) graph-based organization enables us to add new relationships between different types of entities easily. An example is the relationship between a dataset representing training data and the visualization of the corresponding features, which we mentioned in Section 1. More important, the graph allows users to navigate the data-lake catalog in a flexible way and to answer rich queries that require linking datasets to other assets in the enterprise.

4 System Design

GOODS maintains a metadata catalog for the data lake (Figure 1). Our previous work [5] details the physical organization of the catalog, the continuous processes that update it, and its usage to power the user-facing services mentioned in Section 1. Here we highlight some of the important technical approaches we adopted in our system to address the challenges in Section 2.

4.1 Leveraging the Data Lake Relationship Graph

The relationships that we have identified in Section 3 are critical in supporting most of the functionality in GOODS. They enable us both to address some of the scalability challenges that we described in Section 2 and to provide new services to the engineers who use GOODS.

First, clustering provides enormous savings in metadata extraction, albeit potentially at the cost of precision. That is, instead of collecting expensive metadata for each individual dataset, we can collect metadata only for a few datasets in a cluster. We can then propagate the metadata across the other datasets in the cluster. For instance, if the same job generates versions of a dataset daily, these datasets are likely to have the same schema. Thus, we do not need to infer the schema for each version. Similarly, if a user provides a description for a dataset, it usually applies to all members of the cluster and not just the one version. When the clusters are large, the computational savings that we obtain by avoiding analysis of each member of the cluster can be significant.

Second, we can use different types of graph edges to propagate metadata when it is too expensive to extract it directly or we simply do not have this metadata. For instance, we can propagate the description of one version of a dataset to all of its versions. Similarly, versions of the same dataset are likely to have the same schema. Naturally, some of this propagation will introduce uncertainty into our metadata—a fact that we are already dealing with at different levels.

Finally, the links to knowledge graphs representing other parts of Google infrastructure, make GOODS a data-centric starting point for users exploring other resources. For instance, a user can find a definition of a protocol buffer in source code and immediately jump to the list of all the datasets that use that protocol buffer as their schema. A new member of a team can find datasets generated by her team. A profile page for a dataset has links to dataset owners, profile pages for jobs that read and write the dataset, and so on.

4.2 Coordinating Modules for Post-Hoc Processing

The GOODS backend uses a large number of diverse batch-processing jobs to collect information from a variety of systems and to add and update new information into the GOODS catalog. This diversity of jobs is a consequence of our post-hoc approach: we collect information from many diverse corners of the Google's internal infrastructure. Each job includes one or more modules, such as crawlers or analyzers. Different GOODS modules have different characteristics that influence how modules are grouped together in jobs, and how jobs are

scheduled. Next we discuss some of these characteristics and a few rules of thumb that we followed to design and optimize the GOODS backend.

First, not all GOODS modules are critical for the smooth functioning of the system. For example, modules that identify the existence of datasets and extract metadata on who owns and who can access the datasets are critical to ensure that the state of the system is fresh and that any changes in access control for the datasets are reflected accurately. On the other hand, modules like schema analyzer that identifies the schema of a dataset—while useful—are not as time sensitive. Therefore, we explicitly allocate more resources to jobs that include critical modules, and schedule non-critical jobs using spare resources.

Second, certain modules may depend on successful run of other modules. For example, a fingerprint analyzer uses the schema identified by the schema analyzer to compute column level fingerprints. Grouping together dependent modules into the same job enables dependent modules to check the status of the all modules they depend on and take an informed decision. While all GOODS modules store the status of execution for each dataset in the catalog itself to avoid repetitive work on failure and between different instances of the same job, grouping dependent modules reduces the overall number of bytes read from the persistent storage backend.

Third, the failure characteristics of modules vary widely. It is important to isolate some modules which are prone to failure to ensure steady progress of other modules. For instance, several of our modules that examine content of datasets use a variety of libraries specific to different file formats. At times, these libraries crash or go into infinite loops. Because we cannot have long-running analysis jobs crashing or hanging, we sandbox such potentially dangerous jobs in a separate process. We then use a watchdog thread to convert long stalls into crashes while allowing the rest of the pipeline to proceed.

Finally, different modules have different computational complexity and therefore different resource footprints. While we schedule modules that have low complexity to run over the entire catalog every day, we avoid re-running computationally expensive modules unless there is a strong signal that a re-run will yield different results. For example, unless a dataset is modified, the fingerprint for the contents is unlikely to change and therefore the fingerprint analyzer can bypass such dataset.

While some of these design choices were part of our initial design of the GOODS system, we made many of them after experiencing an issue and redesigning parts of our system to address it.

4.3 Search Ranking

As we mentioned in the introduction, dataset discovery through search is the most frequent use case for GOODS. An important design choice for us was to build the search functionality at the level of logical clusters: We index the metadata that describes the logical cluster corresponding to a collection of related datasets (e.g., daily versions of the same dataset) and the user sees results corresponding to these logical datasets. This decision allowed us to compress search results in a meaningful way, instead of overwhelming the user with many similar datasets that match the same query (e.g., showing datasets that differ only on one component of their path denoting their version). We also experimented with the alternative of indexing metadata at the level of each physical dataset, but the end-user experience suffered from the sheer number of similar results.

We faced a few technical difficulties in building the search index. For example, we needed to propagate the metadata from individual members of the cluster itself so that users can search for it. However, by far the biggest challenge was to design a good ranking function for search results. In general, it is hard to overstate the importance of good ranking for search. The problem has unique characteristics in the case of dataset search (and is different than, say, web or bibliographic search) because of the domain-specific signals that can determine the relevance of a dataset to a search query. After much experimentation (and a few false starts), we ended up using a mix of standard IR-type signals (e.g., how well the search terms match the metadata) with domain-specific signals derived from each dataset's metadata. For instance, we found that provenance relationships among datasets provide a strong relevance signal. Specifically, it is common for teams to generate denormalized versions of some "master" dataset in order to facilitate different types of analysis of the master data. These

denormalized datasets can match the same search keywords as the master dataset, yet it is clear that the master dataset should be ranked higher for general-purpose queries or for metadata-extraction. Another example comes from provenance relationships that cross team boundaries, when the dataset from one team is processed to create a dataset in the scope of another team or project. In this case, we can boost the importance of the input dataset as evidenced by its usage by an external team. The output dataset is also important, because we can view it as an origin of other datasets within the external project. Dataset type is another example of a domain-specific signal: for instance our ranking function primes datasets that correpsond to relational databases, because the latter tend to have richer metadata and be more tailored for wide usage compared to, say, files on a distributed filesystem.

Ranking methods for dataset search remains an interesting research problem. It is particularly interesting to consider how techniques from other domains could apply in a data-lake setting, e.g., what would be the equivalent of personalized pagerank when searching for a dataset. However, as we discuss in the next section, our experience with users shows that they have different needs for ranking depending on the purpose of their search. This evidence indicates that a data-lake management system should support several ranking methods and allow users to choose at search time which one to use.

5 Future Directions

After deploying GOODS for the initial use cases that we mentioned in Section 1, we had a chance to observe how the system got adopted, to interact with users, and to receive feedback on feature requests. Based on this information, we have identified a few directions that are interesting for further development on GOODS and, more generally, for the type of functionality that a system for data-lake management can offer.

- **Building a community around datasets** GOODS allows users to share and exchange information about datasets, and to augment the metadata in the catalog with domain-specific knowledge. We view this functionality as the means to develop a community whose shared goal is the curation of the data lake. In this spirit, we can add more community-like features including comments on datasets, reviews, or Q&A functionality, to name a few. The goal is to foster a culture of joint data stewardship and of best practices on adding new datasets to the data lake.

- **Rich analytics over the data lake** The first use case that we targeted with GOODS was dataset discovery within the data lake, through a simple keyword-search interface with relevance ranking. Over time, we found that users "outgrew" this modality and started asking for more flexible ways to query the catalog. The requests ranged from different options for result ranking (e.g., rank datasets by size or by modification timestamp) to full SQL access over the catalog. (In fact, GOODS itself has an internal monitoring component that tracks the state of the data lake through SQL queries over the metadata.) In some cases, users also found it convenient to explore the catalog through a graph visualization based on provenance relationships, which points to the idea of exposing slices of the catalog through specialized user interfaces. Thinking forward, we can also view the catalog as a temporal store that enables comparisons between snapshots of the data lake and thus the discovery of trends. Another option is to view GOODS as a generator of a stream of metadata events, where each event encodes the creation of a dataset, the discovery of a provenance edge, or any other piece of metadata that can be stored in the catalog. Under this model, users can issue continuous queries to monitor the contents of the data lake.. For example, a team can set up a continuous query to monitor accesses to its datasets by other teams, or a user can monitor the generation of datasets by daily runs of some pipeline. In general, our experience with GOODS is that there is value in enabling rich data analytics over the catalog, both to allow users to explore the data lake more effectively but also to monitor the overall state of the data lake.

- **Beyond post-hoc** Our initial approach of crawling and analyzing datasets in a post-hoc manner enabled the collection of basic metadata for datasets stored in many different systems in a minimally invasive

manner. However, there are two main downsides to the post-hoc approach. First, the data in the catalog has a time lag, which conflicts with user expectation of the catalog immediately reflecting the creation or modification of any metadata. Second, the analyzers in GOODS collect only generic, and often uncertain, metadata. As GOODS got adopted within the company, many teams expressed the desire to use GOODS infrastructure to store, retrieve, share, and serve custom metadata for their datasets, ideally within a short time interval of such changes taking place. In order to tackle such use cases, we envision a hybrid approach that supports one-off deeper integration with storage infrastructures to reduce the time lag for discovery, and APIs for teams to register datasets and to contribute custom metadata to the catalog.

- **Acting on data** The information in the catalog not only helps users understand the datasets that they know about but also enables them to discover what they can do with their datasets, how they can use datasets in other tools, or to discover new related datasets. More concretely, one of the very popular features of profile pages in GOODS are pre-populated queries and code snippets that use the path and schema information, which users can simply copy and paste into other tools. We can envision extensions of this feature that are based on a deeper analysis of the dataset's metadata and that can help the user act on the dataset with more elaborate tools; for example, we can automatically build a dashboard for the dataset based on its characteristics. These extensions can become more powerful if we can leverage the relationships encoded in the catalog. For instance, if GOODS can tell us that a key column in our dataset has very similar context to a key column in another dataset, then the two datasets might be candidates for joining and the user may even be presented with possible actions on the join results. This specific example is intriguing, as the data-lake management system helps the user understand what datasets *could* exist in the data lake instead of merely summarizing what has already been generated.

Overall, a data-lake management system should promote the treatment of datasets as first-class objects within the computing infrastructure of the enteprise. A big part of this goal involves services that make it easier for engineers and analysts to integrate datasets within their workflow in an organic fashion, and this direction has been the main focus of our work with GOODS. Furthermore, through these services the system can instill best practices for dataset management and break team-delinated silos, thus fostering a culture of responsible data stewardship across the enterprise.

References

[1] Azure data lake. `https://azure.microsoft.com/en-us/solutions/data-lake/`.

[2] Data lakes and the promise of unsiloed data. `http://www.pwc.com/us/en/technology-forecast/2014/cloud-computing/features/data-lakes.html`.

[3] A. Brown. Get smarter answers from the knowledge graph. `http://insidesearch.blogspot.com/2012/12/get-smarter-answers-from-knowledge_4.html`, 2012.

[4] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26(2):4:1–4:26, June 2008.

[5] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang. Goods: Organizing google's datasets. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 795–806, New York, NY, USA, 2016. ACM.

[6] I. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino. Data wrangling: The challenging journey from the wild to the lake. In *CIDR 2015, Seventh Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA*, 2015.

[7] K. Varda. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Accessed July*, 2008.