2018 IEEE 14th International Conference on e-Science

# Skluma: An extensible metadata extraction pipeline for disorganized data

Tyler J. Skluzacek*, Rohan Kumar*, Ryan Chard‡, Galen Harrison*,
Paul Beckman*, Kyle Chard†‡, and Ian T. Foster*†‡
*Department of Computer Science, University of Chicago, Chicago, IL, USA
†Globus, University of Chicago, Chicago, IL, USA
‡Data Science and Learning Division, Argonne National Laboratory, Argonne, IL, USA

***Abstract*—To mitigate the effects of high-velocity data expansion and to automate the organization of filesystems and data repositories, we have developed Skluma—a system that automatically processes a target filesystem or repository, extracts content- and context-based metadata, and organizes extracted metadata for subsequent use. Skluma is able to extract diverse metadata, including aggregate values derived from embedded structured data; named entities and latent topics buried within free-text documents; and content encoded in images. Skluma implements an overarching probabilistic pipeline to extract increasingly specific metadata from files. It applies machine learning methods to determine file types, dynamically prioritizes and then executes a suite of metadata extractors, and explores contextual metadata based on relationships among files. The derived metadata, represented in JSON, describes probabilistic knowledge of each file that may be subsequently used for discovery or organization. Skluma's architecture enables it to be deployed both locally and used as an on-demand, cloud-hosted service to create and execute dynamic extraction workflows on massive numbers of files. It is modular and extensible—allowing users to contribute their own specialized metadata extractors. Thus far we have tested Skluma on local filesystems, remote FTP-accessible servers, and publicly-accessible Globus endpoints. We have demonstrated its efficacy by applying it to a scientific environmental data repository of more than 500,000 files. We show that we can extract metadata from those files with modest cloud costs in a few hours.**

***Index Terms*—Metadata extraction, data swamp**

## I. Introduction

Scientists have grown accustomed to instantly finding information, for example papers relevant to their research or scientific facts needed for their experiments. However, the same is not often true for scientific data. Irrespective of where data are stored (e.g., data repositories, file systems, cloud-based object stores) it is often remarkably difficult to discover and understand scientific data. This lack of data accessibility manifests itself in several ways: it results in unnecessary overheads on scientists as significant time is spent wrangling data [1]; it affects reproducibility as important data (and the methods by which they were obtained) cannot be found; and ultimately it impedes scientific discovery. In the rush to conduct experiments, generate datasets, and analyze results, it is easy to overlook the best practices that ensure that data retain their usefulness and value.

We [2], [3], and others [4], [5], [6], [7], have developed systems that provide metadata catalogs to support the organization and discovery of research data. However, these approaches require upfront effort to enter metadata and ongoing maintenance effort by curators. Without significant upkeep, scientific data repositories and file systems often become *data swamps*—collections of data that cannot be usefully reused without significant manual effort [8]. Making sense of a data swamp requires that users crawl through vast amounts of data, parse cryptic file names, decompress archived file formats, identify file schemas, impute missing headers, demarcate encoded null values, trawl through text to identify locations or understand data, and examine images for particular content. As scientific repositories scale beyond human-manageable levels, increasingly exceeding several petabytes and billions of individual files, automated methods are needed to *drain* the data swamp: that is, to extract descriptive metadata, organize those metadata for human and machine accessibility, and provide interfaces via which data can be quickly discovered.

While automated methods exist for extracting and indexing metadata from personal and enterprise data [9], [10], such solutions do not exist for scientific data, perhaps due to the complexity and specificity of that environment. For example, many of the myriad scientific data formats are used by only a small community of users—but are vital to those communities. Furthermore, even standard formats adopted by large communities are often used in proprietary and ad hoc manners.

In response, we have developed Skluma [11], a modular, scalable system for extracting metadata from scientific files and collating those metadata so that they can be used for discovery across repositories and file systems. Skluma applies a collection of specialized "metadata extractors" to files. It is able to crawl files in many commonly-used repository-types (e.g., object stores, local file systems) and accessible via various access protocols (e.g., FTP, HTTP, Globus). Skluma dynamically constructs a pipeline in which progressively more focused extractors are applied to different files and to different parts of each file. It automatically determines which metadata extractors are most likely to yield valuable metadata from a file and adapts the processing pipeline based on what is learned from each extractor.

We evaluate Skluma's capabilities using a real-world scientific repository: the Carbon Dioxide Information Analysis Center (CDIAC) dataset. CDIAC is publicly available, containing 500,001 files ranging from tabular scientific data; photograph,

978-1-5386-9156-4/18/$31.00 ©2018 IEEE
DOI 10.1109/eScience.2018.00040

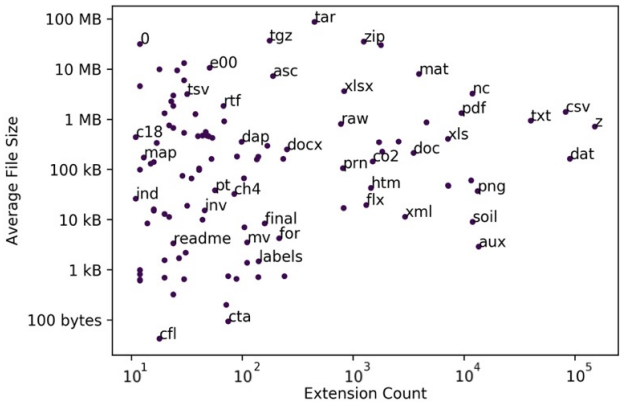



Fig. 1: Average file size vs. counts of common file extensions found in the CDIAC data lake; 500,001 total files (excluding files with counts $< 10$).

map, and plot images; READMEs, papers, and abstracts; and a number of scientifically uninteresting files (e.g., Hadoop error logs, Windows installers, desktop shortcuts). CDIAC contains 152 file extensions, as shown in Figure 1. We show that we can effectively extract metadata, with near-linear scalability.

The remainder of this paper is as follows. Section II formalizes the problem motivating Skluma and introduces the files that are targeted by Skluma's workflows. Section III outlines Skluma's overall architecture. Section IV discusses Skluma's built-in extractors. Section V evaluates Skluma from the perspectives of performance, correctness, and usability. Finally, Sections VI and VII discuss related efforts, future work, and conclusions.

## II. Problem description

Skluma performs end-to-end metadata extraction using a series of encapsulated tasks called extractors. These extractors are applied to files within a repository to derive structured property sets of metadata. We formalize this as follows.

A **repository** $R$ is a collection of files, with each $f \in R$ comprising a file system path and a sequence of bytes, $f.p$ and $f.b$. A **property set** $M$ is a collection of metadata, with each $m \in M$ comprising a file system path, a metadata element, source extractor, and timestamp, $m.p$, $m.e$, $m.s$, and $m.t$, respectively. A property set $M$ is said to be **valid** for a repository $R$ if for each $m \in M$, there is an $f \in F$ such that $m.p = f.p$. The metadata associated with a file $f \in F$ are then $M_f = m \in M : m.p = f.p$.

We have a set of **extractors**, $\mathcal{E}$. An extractor $e \in \mathcal{E}$ is a function $e(f, M_f)$ that returns a (potentially empty) set of new metadata elements, $N$, such that for each $m \in N$, $m.p = f.p$ and $m.s = e$.

The metadata extraction process for a file $f$ proceeds by: (i) calling a function **next**$(M_f, h_f)$, $h_f$ being the (initially empty) sequence of extractors applied to $f$ so far, to determine the extractor $e \in E$ to apply next; (ii) evaluating $N = e(f, M_f)$; and (iii) adding $N$ to $M$ and $e$ to $h_f$. These steps are repeated until **next**$(M_f, h_f) = \phi$.

Note that in this formulation of the metadata extraction process, each extractor $e$ is only provided with access to metadata, $M_f$, about the specific file $f$ that it is processing: thus information learned about one file cannot directly inform extraction for another. This restriction is important for performance reasons. However, as we discuss in the following, we find it helpful for both extractors and the **next** function to be able to access models created by analysis of metadata extracted from previously processed files.

### A. File Types

Scientific repositories and file systems contain a wide range of file types, from free text through to images. The methods by which meaningful metadata can be extracted from file types differs based on the representation and encoding. Thus far we have isolated a number of prevalent scientific file type classes and have built their corresponding extractors. While recognizing that there are hundreds of well-known file formats [12], we have thus far focused efforts on the following file type classes:

**Unstructured:** Files containing free text. Often human-readable natural language and encoded in ASCII or Unicode. Valuable metadata are generally related to the semantic meaning of the text, such as topics, keywords, places, etc. Unstructured files are often stored as *.txt* or *README* files.

**Structured:** Encoded or organized containers of data in a predefined file format. Often includes self-describing metadata that can be extracted through standard tools and interfaces. Common formats include *JSON*, *XML*, *HDF* and *NetCDF* files. Structured files may contain encoded null values.

**Tabular:** Files containing tabular data that are formatted in rows and columns and often include a header of column labels. Metadata can be derived from the header, rows, or columns. Aggregate column-level metadata (e.g., averages and maximums) often provide useful insights. Structured files are often stored as *.csv* and *.tsv* files.

**Images:** Files containing graphical images, such as plots, maps, and photographs. Common formats include *.png*, *.jpg*, and *.tif*.

**Compressed:** This class encompasses any file type that can be decompressed with known decompression software to yield one or more new files. Example extensions are *.zip* and *.gz*. The new file(s) may or may not be of types listed here.

**Other:** Files that are not recognized as belonging to one of the types listed above. In the CDIAC repository, we find Microsoft *.docx* and *.xlsx* files, for example.

**Hybrid:** Files containing multiple data types, such as tabular data with free text descriptions and images with text labels.

## III. Skluma architecture

Skluma is designed as a web service, with a JSON-based REST API that users can use to request metadata extraction from either an entire repository or a single file. An extraction request defines the repository or file to be processed, the

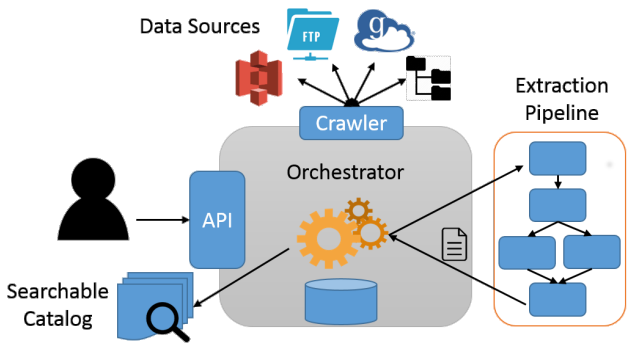



Fig. 2: Skluma architecture showing the crawler used to traverse various file systems, the orchestrator for coordinating processing, an extraction pipeline composed of extractors for processing files, and the catalog containing extracted metadata.

crawler to use, optional configuration settings, and any custom extractors to be considered in the extraction process.

Skluma (see Figure 2) comprises three core components: a scalable **crawler** for processing target repositories, a set of **extractors** for deriving metadata from files, and an **orchestrator** that implements the API, invokes crawlers, manages metadata extraction for each file identified by a crawler, and manages the resources used by crawlers and extractors. We provide details on each component in the following.

The orchestrator, the crawler, and the different extractors are each packaged in separate Docker containers. This architecture makes scaling trivial. The crawler can run concurrently with extractors, and additional container instances can be created as needed to crawl multiple repositories at once and/or to extract metadata from multiple files at once. A well-defined extractor interface enables modularity and extensibility.

Skluma can be deployed both locally and in the cloud. In the first case, the entire Skluma stack is deployed locally and containers are executed on the host. In the cloud-based model, the Skluma service is hosted in one cloud instance, and crawler and extractor instances are executed in single-tenant cloud containers.

In both local and cloud deployments, users configure the maximum number of concurrent container instances to be run, currently $n \geq 3$ (1+ crawlers, 1 orchestrator, and 1+ extractors). In local deployments, Skluma manages container execution using Docker. In cloud deployments, Skluma uses the Amazon Web Services (AWS) Elastic Container Service (ECS) to manage the allocation of compute resources and the mapping of container instances onto those compute resources.

Skluma is designed to address several requirements: extensibility and modularity to enable customization for different repositories and domains (e.g., via implementation of new crawlers or extractors), scalability and performance to support the processing of large repositories, smart and adaptive extraction pipelines that learn and guide extraction decisions, and comprehensibility to capture a diverse set of metadata with different levels of accuracy and value.

### A. The Crawler

Skluma can be applied to a wide range of data stores, from local file systems through to web-based data repositories. To this end, we define a modular data access interface with functions for retrieving file system metadata, accessing file contents, and recursively listing folder contents. We provide implementations of this interface for different storage systems: so far, HTTP, FTP, Globus, and POSIX.

The orchestrator invokes the crawler in response to an extraction request. The crawler is launched within a container to process an entire repository. Starting from one or more root locations, it expands directories recursively and records each file that it encounters in a database as a (unique identifier, file system path) pair.

If multiple repositories need to be crawled, the orchestrator can launch multiple crawler tasks at once, each within its own container.

### B. Extractors

An extractor is an application that, when applied to a file plus a set of metadata for that file, generates zero or more additional metadata items. Most extractors are type-specific. For example, they may be designed to extract values from a tabular file, content from an image, or semantic meaning from text. Here we introduce two preparatory extractors, **`universal`** and **`sampler`**; the other seven extractors that we have developed to date are described in Section IV.

The **`universal`** extractor extracts file system metadata—(e.g., name, path, size, extension)—and computes a file checksum that can be used to identify duplicate files. When the file extension suggests a known compression format (e.g., *.gz*, *.zip*), it attempts to decompress the file with the appropriate decompressor, and upon success adds the resulting file(s) to the set of files to be processed.

The **`sampler`** extractor reads selected file contents and uses that information, plus file system metadata extracted by **`universal`**, to provide an initial guess at file type. For example, if extracted text are all ASCII and the filename prefix is `.txt`, **`sampler`** may infer that a file is a text file.

Subsequent extractors applied to a file vary according to both the file's inferred type and extracted metadata. For example, if **`sampler`** infers a file type of "text," then Skluma's **`next`** function may next select a text extractor. If the file furthermore contains both lines representing tabular data (e.g., consistent delimiters and uniform inferred column types) and a free text header (e.g., describing the project or experiment), the file will also be processed by tabular and free text extractors. If a tabular extractor identifies a high probability of null values (e.g., repeated outliers or non-matching types), Skluma will apply a null value extractor to the file. Thus two files of the same type, as inferred by **`sampler`**, can have quite different extraction pipelines.

An extractor is implemented, either by itself or colocated with additional extractors, in a Docker container with a common interface. This organization provides for modularity, extensibility, enables trivial scaling, and simplifies execution

in heterogeneous computing environments [13]. Extractors are registered with Skluma through a simple JSON configuration that defines the unique extractor name (i.e., container name used for invocation) and hints about the file types that it can be used to process. Extractors must implement a simple interface that accepts three parameters: a unique resource identifier (URI) to a file (e.g., the path to a file in a Globus endpoint) and that file's current metadata (JSON). An extractor returns either a (perhaps empty) set of extracted metadata or an error.

Extracted metadata are added to set of metadata associated with the file, with, as noted in Section II, the extractor name appended to indicate the metadata's source. This approach allows for disambiguation when multiple extractors assert the same value.

To improve performance by enabling data reuse, extractors can be co-located within a single container. This is especially beneficial in the case of extractors that are often applied to the same file. For example, co-locating the free text **`topic`** extractor with the **`tabular`** data extractor can prove useful in situations where tabular files include free text preambles.

### C. The Orchestrator

The orchestrator manages the end-to-end extraction process. It implements the API by which users submit, and monitor the progress of, extraction tasks. For each such task, it is responsible for invoking the crawler on the repository named in the request, coordinating metadata extraction for each file identified by the crawler, and adding the extracted metadata to the metadata database. It is also responsible for managing the pool of resources used by crawlers and extractors.

The orchestrator processes each file identified by the crawler by repeatedly calling the **`next`** function (see Section II) to determine the next extractor and then running that extractor on the file and its metadata, until the **`next`** function indicates that there are no more extractors to run. (The extraction process for a file currently terminates when the last extractor fails to identify future extraction steps for **`next`**. In the future, we may also consider other factors, such as a limit on the resources to be spent extracting metadata.) The sequence of extractors thus applied to a file constitutes the file's **extraction pipeline**.

Skluma extractors can, in principle, be applied in any order. However, we will typically see **`universal`** applied first, to extract file system metadata and to process compressed files, then **`sampler`** to infer file type, and then various increasingly specific extractors depending on that type.

### D. File Updates

The constant churn of scientific file systems and repositories makes static indexing unfeasible. For a metadata catalog to be useful its contents must be up-to-date, reflecting changes to files as they are added, copied, or edited. Techniques to dynamically construct, update, and curate metadata catalogs are required for them to provide the most use to researchers. To facilitate online metadata extraction and enable Skluma to update itself and its catalogs in real-time, we use the Ripple [14] event detection service. When the file is initially added to the system, it gets an entirely new metadata document. When that same file is edited, the file is reprocessed entirely, swapping the old metadata for the new. We do not track a file's version changes at this time.

### E. Output Metadata

The output of the pipeline is a metadata document for each processed file, as shown in Listing 1: a hierarchical JSON document that contains both file system metadata (ownership, modification time, and path) and the information generated by each extractor. Each extractor will generate domain-specific metadata from a given file. For example, the tabular data extractor, commonly applied to comma-separated-value files, will create a list of header descriptions and aggregate, column-level information (e.g., type; min, max, average values; precision; and inferred null values).

One important use for extracted metadata is to implement a search index that allows flexible search across files in a repository. Thus, we commonly export metadata to Globus Search—a scalable cloud-hosted search service with fine grain access control and user managed indexes [15]. To do so, Skluma translates the metadata output into the GMeta format required by Globus Search and, via the Globus Search API, pushes the metadata to a user-defined index.

## IV. Extractors

Given the tremendously wide range of scientific file formats, we have developed a modular architecture that allows user-defined extractors to be plugged in easily. We introduced the **`universal`** extractor in Section III-B; here we describe the **`sampler`** extractor and also extractors for tabular, text, and image formats.

### A. The *`sampler`* extractor

Applying every Skluma extractor to every file would be both time consuming and unnecessary: for example, there is little insight to be gained from applying an image extractor to structured text. In order to identify the most suitable extractor(s) to apply to each file, we use the **`sampler`** extractor to sample and predict each file's content. This extractor uses the metadata captured by **`universal`** in combination with a (configurable) sample of $N$ bytes in a file. The sample is then converted into features that are used by a machine learning model to predict a file's type. The file's type is used to bootstrap the selection of which other extractors should be applied to the file.

Any preprocessing done during the sampling process can be costly, for example to determine fields, delimiters, or even lines. To avoid such preprocessing, we compute byte $n$-grams from the sampled content and train our selected models to determine the file type [16]. We explored various size $n$-grams and found that anything greater than $n = 1$ was prohibitively slow with negligible improvements to accuracy. We also explored two ways of sampling the file: *head* (all from top of the file) and *randhead* (half from top of file, half from randomly throughout the rest of the file). We explore the performance of these approaches in Section V.

Listing 1: Example Skluma metadata for a hybrid precipitation data file containing a free text header and structured data; processed on a local machine. Universal, sampler, tabular, and topic extractors operate on the file.

```
"metadata":{
  "file":{
    "URI": "local::/home/skluzacek/data/precip08.csv",
    "updated":"03/08/2018-15:08:4537",
    "owner":"skluzacek"
  }
  "extractors":{
    "ex_universal":{
      "timestamp":"03/08/2018-15:25:1233",
      "name":"precip08",
      "ext":"csv",
      "size":"3091287"
    },
    "ex_sampler":{
      "timestamp":"03/08/2018-15:26:3163",
      "tabular-header"
    },
    "ex_tabular":{
      "timestamp":"03/08/2018-15:27:5390",
      "headers":["id", "latx", "date"],
      "cols":{
        "id":{
          "type":"str"
        },
        "latx":{
          "type":"latstd-float",
          "prec":".1",
          "null":"-999",
          "min": "0.0",
          "max":"180.0",
          "avg":"91.5"
        },
        "date":{
          "type":"datetime",
          "early":"02/04/2014",
          "late":"06/06/2014"
        }
      }
    },
    "ex_topic":{
      "timestamp":"03/08/2018-15:27:6340",
      "type":"preamble",
      "keywords":[("precipitation","0.8"), ("transpiration"
          ,"0.6"), ("aerosols", "0.4"), ("UChicago","0.2")]
          ,
      "topics":["atmospheric science", "climate science", "
          oceanography", "meteorology", "weather"]
    }
  }
}
```

There are many machine learning models that perform well in supervised classification problems including Support Vector Machines (SVMs), logistic regression, and random forests, however all require "ground truth" labels for training. As all repositories are different and users may provide their own extractors, we have developed an approach to support training on a subset of the repository. Skluma randomly selects a configurable percentage (e.g., $k = 20\%$) of files from a given repository. It will then run these files through the pool of extractors and determine for which extractors metadata are successfully extracted (i.e., when a given extractor does not throw an *ExtractionError*). Once $k\%$ of the files are processed, Skluma trains a model on these results and the model is stored in the **`sampler`** container's data volume for future prediction.

### B. Content-Specific Extractors

Having identified the likely file type, the next stage of the pipeline focuses on content-specific extraction. We have developed an initial set of extractors designed for common scientific file types. Although these extractors target specific file types, they are not mutually exclusive. For example Skluma may apply a tabular extractor to one part of a file and also apply a topic extractor to another part of that file. The remainder of this section outlines several Skluma extractors. These extractors are summarized in Table I.

*1) Tabular:* Scientific data are often encoded in tabular, column-formatted forms, for example, in comma-separated-value files. Such data might appear trivial to parse and analyze. However, many scientific datasets do not conform strictly to such simple specifications. For example, we found in CDIAC many column-formatted data files with arbitrary free text descriptions of the data, multiple distinct tabular sections (with different header rows and number of columns), missing header rows, and inconsistent delimiters and null value definitions.

Skluma's **`tabular`** data extractor is designed to identify and extract column-formatted data within text files. Its implementation applies various heuristics to account for the challenges described above. For example, if the first line of a file is not recognizable as a header row (e.g., it has a different number of columns to the subsequent lines), the extractor initiates a binary search over the file to identify the start of the delimited data. The extractor also works to identify columns (e.g., by identifying delimiters and line-break symbols) and to derive context (e.g., name, data type) and aggregate values (e.g., the mode, min, max, precision, and average for a numerical column; the bounding box for a geospatial column; and the $n$ most common strings in a text column).

Computing aggregate values is not without risk. For example, we sometimes find ad hoc representations (e.g., empty string, "null," or an unrealistic number such as $-999$) used to encode missing values. Without knowledge of such schemes, aggregate values can be meaningless. Furthermore, detecting their use can require domain knowledge: for example, $-99$ degrees is an unrealistic temperature for Chicago, but not for Neptune. To capture such patterns we have developed a **`null`** inference extractor that employs a supervised learning model to determine null values so that they can be excluded from aggregate calculations. We use a $k$-nearest neighbor classification algorithm, using the average, the mode, the three largest values and their differences, and the three smallest values and their differences as features for our model. By taking a classification rather than regression-based approach, we select from a preset list of null values, which avoids discounting real experimental outliers recorded in the data.

*2) Structured:* Structured files contain data that can be parsed into fields and their respective values. Extracting this data line-by-line proves ineffective due to each employing a specialized file schema. We have developed the **`structured`** extractor to decode and retrieve metadata from two broad classes of structured files: data containers and documents.

TABLE I: Skluma's current extractor library. (Extensions listed are just examples.)

| Name | File Type | Extensions | Brief Description |
|---|---|---|---|
| **universal** | All | All | Compute file hash and record file system information (size, name, extension) |
| **sampler** | All | All | Generate hints to guide extractor selection |
| **tabular** | Column-structured | CSV, TSV | Column-level metadata and aggregates, null and header analyses |
| **null** | Structured | CSV, TSV | Compute single-class nulls for each column |
| **structured** | Hierarchical | HDF, NetCDF | Extract headings and compute attribute-level metadata |
| **topic** | Unstructured | Txt, Readme | Extract keywords and topic tags |
| **image** | Image | PNG, TIF | SVM analysis to determine image type (map, plot, photograph, etc.) |
| **plot** | Image | PNG, TIF | Extract plot titles, labels, and captions; assign axis-ranges and keywords |
| **map** | Image | PNG, TIF | Extract geographical coordinates of maps; assign coordinates and area-codes |

**structured** first uses a simple set of rules to determine in which of the two classes the file belongs.

Containerized data formats (e.g., HDF, NetCDF) encode metadata in standard formats that are accessible via standard interfaces. Once a file is determined to be containerized, the **structured** extractor first uses a second set of rules to determine the container type (e.g., HDF, NetCDF) and then calls appropriate container-specific libraries to retrieve header values and in-file self-describing metadata, and to compute a series of aggregate values for each dimension.

Structured document formats (e.g., XML, JSON, etc.) contain a number of key-value pairs. For scientific data, these keys can be considered "headers" and the values corresponding data. **structured** automatically parses the keys from the file and computes aggregate values for each key containing numerical lists, while also storing the nonnumerical keys as a metadata attribute list.

*3) Unstructured:* Files in scientific repositories often contain free text (mostly, but certainly not always, in English). Text may be found, for example, in purely or primarily textual documents (e.g., READMEs, paper abstracts) or within certain fields of a structured document. Purely textual documents may be intended as documentation for other files, as sources of data in their own right, or for other purposes. In future work, we intend to infer relationships among such documents and associated files and to investigate methods for extracting relevant data, via automated and/or crowdsourcing methods [17]. For now, however, our **topic** metadata extractor is designed to process free text within files. It takes a blob of free text as an input, and derives keyword and topic labels. We consider three types of free text extraction: structured text extractors, latent keyword extractors, and topic models.

Structured extractors allow the **topic** extractor to extract domain-specific metadata. In the case of CDIAC, geographical attributes such as geological formations, landmarks, countries, and cities are potentially valuable metadata attributes, but are often hidden deep within files. To provide this capability we have extended the Geograpy [18] text-mining Python library. This library is representative of text extraction processes using a well-known ontology or vocabulary of attributes to be extracted. The output of the model is an unranked list of geographical attributes, including both man-made and natural landmarks, cities, states, and countries.

Many important metadata attributes do not map to a published ontology or vocabulary. In this case, more general latent keyword extraction can provide meaningful metadata. Here, we apply a multi-stage approach, first using a simple, non-probabilistic model to identify keywords and then an additional topic mixture model to find related topics.

While our keyword extraction method can extract words within the document, it often does not provide sufficient semantic information about a document (e.g., related to words not explicitly included in the text). Thus we leverage word embeddings—specifically, pre-trained word embeddings from the GloVe project [19]—as our second stage [20]. Using the corresponding vectors of the top-$n$ keywords, we compute an aggregate vector to represent each document, weighted by the amount of information they provide (as above). Since each document is now represented by a vector in the space of word embeddings, it can be used as a search facet when searching via topics, as one can easily retrieve the closest words to that vector representing a topic. More specifically, any query topic string could be converted to one or more vectors in this space, and the search could return those documents which are semantically closest to these query vectors. Another benefit of having document vectors in a space that preserves semantic relationships is that documents can be compared using their vectors to ascertain the similarity between their main topics.

**topic** also uses a topic modeling approach to extract contextual topics that provide even more information when combined with a text file's keywords. For this purpose, we use a topic mixture model based on Latent Dirichlet Allocation (LDA) [21]. We trained our topic model on Web of Science (WoS) abstracts (as a proxy for general scientific text). We can then use the resulting model to generate topic distribution for free-text files: the finite mixture over an underlying set of topics derived from the model.

This LDA approach works well on longer text files. It works less well for the many small text files (e.g., READMEs) contained in the CDIAC repository. We hypothesize that this is because these documents are neither long enough nor sufficiently well structured for LDA sampling to extract coherent topic models.

*4) Images:* Scientific repositories often contain large numbers of image files, such as graphs, charts, photographs, and maps. These files often contain useful structured metadata, such as color palette, size, and resolution, that can be extracted easily given knowledge of the file format. Other attributes important for discovery relate to the content represented by the image: for example, the locations in a map or the anatomical region displayed in an X-ray image. To extract those attributes, we first work to identify the type of the image (e.g., map or

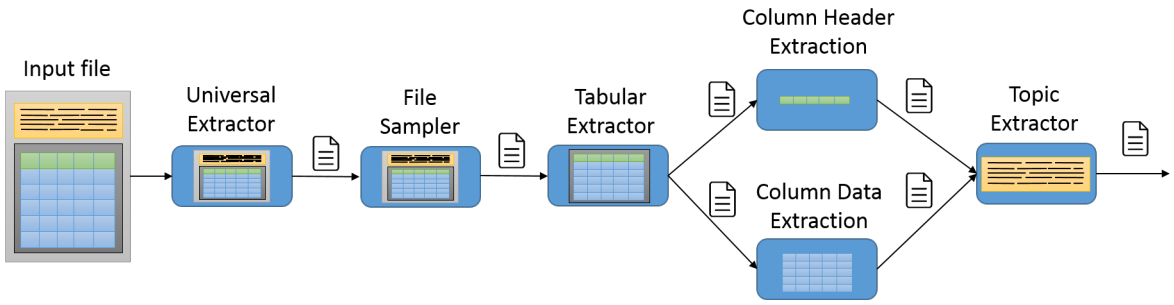



Fig. 3: Progression of metadata extraction tasks for example structured-unstructured hybrid file.

x-ray) and then apply more sophisticated extraction methods.

We first apply a specialized **`image`** extractor that classifies image types. This extractor resizes an image to standard dimensions, converts it to grayscale, and passes the result to a Support Vector Machine (SVM) model that returns a most likely class: plot, map, photograph, etc. We train the SVM model offline with a set of labeled training images.

Having inferred a likely image type, Skluma then applies one or more specialized extractors: for example **`plot`** to extract titles from plots and **`map`** to extract locations for maps.

The **`plot`** extractor applies the Tesseract optical character recognition (OCR) engine [22] to identify and extract text from the image. At present these text fragments are returned without regard for where they appear in the image.

The **`map`** extractor attempts to derive meaningful geolocation metadata from images. It first extracts text present in the image by applying a sequence of morphological filters to the image, using a contour-finding algorithm to identify contours [23], and filtering contours to identify text characters, which it then groups by location to form text boxes [24], as shown in Figure 4. Each text box is then passed through Tesseract to extract text. Each valid piece of text is parsed either as a latitude or longitude coordinate, or as the name of a location. If at least two pairs of valid coordinates are found, a pixel-to-coordinates map is created for later use.

The **`map`** extractor next applies a different set of morphological filters to the original image for border extraction. Contours are found by the same algorithm above and are filtered to isolate those that are large enough to be borders of geographical regions. Note that this approach assumes that any geographical regions of significance will not be negligibly small relative to the size of the image. The pixel-to-coordinates map transforms each contour into a set of (latitude, longitude) pairs bordering a region. Each coordinate-border is searched within an index of regional shapefiles to find any geographical regions with which it overlaps (see Figure 4). If the border extraction fails, we extract at least the (latitude, longitude) span of the map, which is useful in its own right, although less specific [25]. At this point in the pipeline, we have extracted the latitudes and longitudes that the map spans, and we have tagged each substantial region found in the map with the geographical location(s) that it covers.

These contour-finding, text-isolation, and text-extraction techniques can also be extended to other sorts of images. For example, we could use contours to identify and characterize scientific plots within an image (e.g., histograms, pie-charts), and perhaps even to extract the data that they represent.

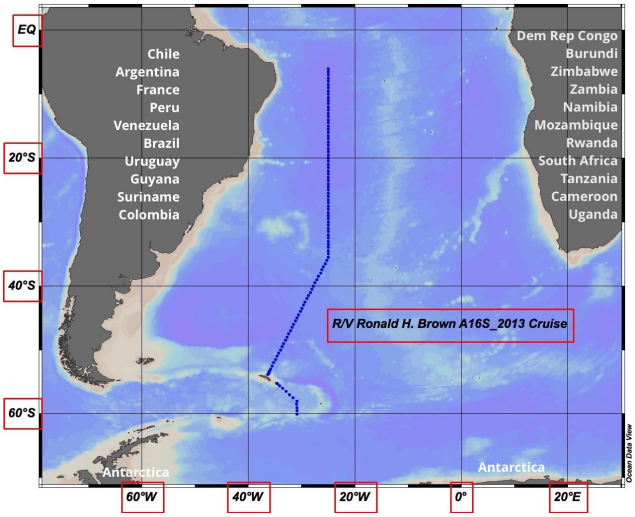



Fig. 4: An example of location metadata extraction from maps. The red boxes delineate the extracted text; the white text is the tags generated for each region found in the map.

## V. Evaluation

We evaluate Skluma's accuracy and performance over a number of metadata indexing tasks. We process the 500,001 file (303 GB) CDIAC repository. CDIAC contains a large collection of minimally curated scientific data including an assortment of images, structured data files, hierarchical files, encoded files, and unstructured files: see Table II. We specifically evaluate the accuracy of the file sampler, Skluma's overall correctness, and the performance of both cloud and local deployments of Skluma. Finally, we project the costs necessary to index larger scientific stores.

### A. Extractor accuracy on CDIAC data

We first explore the accuracy of the **`sampler`**, **`topic`**, **`null`**, and **`map`** extractors.

TABLE II: Distribution of file types, and median sizes per type, in the CDIAC repository.

| File type | Count | Size (KB) |
|---|---|---|
| Structured | 22,127 | 11.90 |
| Unstructured | 53,184 | 13.57 |
| Tabular | 173,019 | 12.90 |
| Image | 32,127 | 118.20 |
| Specialized | 156,191 | 20.14 |
| Other | 63,353 | 15.32 |
| **Total** | **500,001** | **15.41** |

TABLE III: Model accuracy and training and reading times, for 20,000 randomly selected CDIAC files.

| Model | Features | Accuracy (%) | Time (sec) Train | Time (sec) Read |
|---|---|---|---|---|
| RandForest | Head | 93.3 | 1.77 | 3.69 |
| | RandHead | 90.6 | 3.76 | 356.40 |
| Logistic | Head | 92.4 | 597.00 | 2.97 |
| | Random | 83.9 | 958.00 | 290.00 |
| Baseline | | 71.4 | 0.00 | 9,097.00 |

*1) Accuracy of the* **`sampler`** *extractor:* We explore here the accuracy and training time when using two machine learning models (random forests: *RandForest*, or logistic regression: *Logistic*) and two sampling strategies (from the head of the file: *Head*, or half each from the head and random locations in the file: *RandHead*). We also include a baseline model that uses labels obtained by running the tabular, structured, free text, and image extractors in sequence until either (1) one of the four successfully extracts metadata, or (2) all four raise exceptions.

We begin by selecting 20,000 CDIAC files uniformly at random from the index. We use five-fold cross validation by randomly shuffling the files into five groups where each group serves as the training set for the other four (each withholding type labels) exactly once. Accuracy is measured as the percentage of successfully labeled files in each testing subset. Table III reports the accuracy and training time for the feature sets and models. We see that both models are significantly more accurate than the baseline approach. Random forests performs better than logistic regression, but only slightly. Sampling the head of the file, rather than using the head and a random sample throughout the file, is more accurate for both models, which is fortunate as the latter takes much longer due to the need to scan through the file.

We note that time spent training this model is negligible (we can train a random forests model using the head of the file in just over three seconds) relative to the cost of running extractors (see average individual extractor times in Table IV).

*2) Accuracy of the* **`topic`** *extractor:* We employ human reviewers to validate extracted topics and keywords. We measure accuracy as the percentage of files in which the extracted topics and keywords provide correct, useful information. Topics are considered *correct* if they match the intended topic of the text. Topics are considered *useful* if they subjectively reflect the content of a file. We introduce three mutually exclusive classes to determine accuracy: (1) correct and useful, (2) incorrect, (3) not useful. Incorrect metadata include incorrect terms: for example, a README about rainforest precipitation labeled as 'concrete.' Non-useful metadata are correct but too vague: for example, a PDF about jetstream currents labeled as 'science.' We tasked our human reviewers with assigning each file to one of the three classes. They examined 250 files with extracted topic metadata, and deemed those metadata to be correct and useful for 236, not useful for 14, and incorrect for none, for an accuracy of 94%.

*3) Accuracy of the* **`null`** *extractor:* We measure the correctness of the null value extractor by comparing the output of columns processed by the extractor with each column's human-labeled null values. When trained and tested by cross-validation on a labeled test set of 4,682 columns from 335 unique files, our model achieved accuracy 99%, precision 99%, and recall 96%, where both precision and recall are calculated by macro-averaging over all classifiers.

*4) Accuracy of the* **`map`** *extractor:* We selected 250 map images at random from the 583 files classified as maps, used **`map`** to assign metadata to each of them, and asked human reviewers to classify the metadata into the three classes listed above: correct and useful, incorrect, or not useful. Our reviewers found that the metadata were correct and useful for 231, incorrect for 7, and not useful for 12, for an accuracy of 92%. Two reasons for incorrect metadata were uncommon projections and/or fuzzy text. Thus, we plan to extend the **`map`** extractor to account for additional projections and to provide a confidence metric for extracted text.

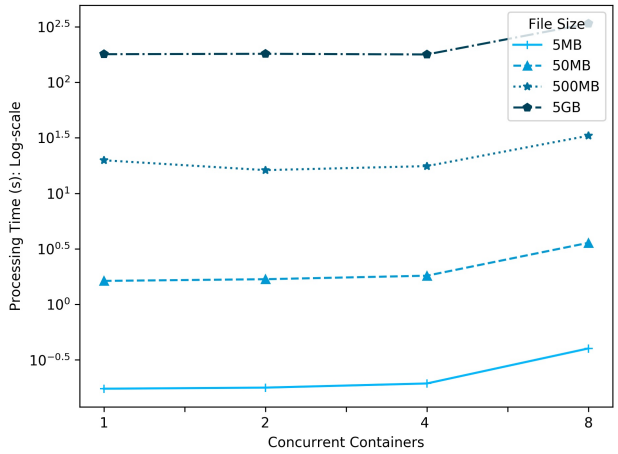
### *B. Performance*

We evaluate Skluma's performance in four ways: first, we investigate how quickly the crawler can perform a depth-first search of a repository; second, we investigate extractor performance with respect to input size and extractor concurrency; third, we analyze the performance of each extractor as they are used to process the entire CDIAC repository; finally, we review the entire indexing pipeline.

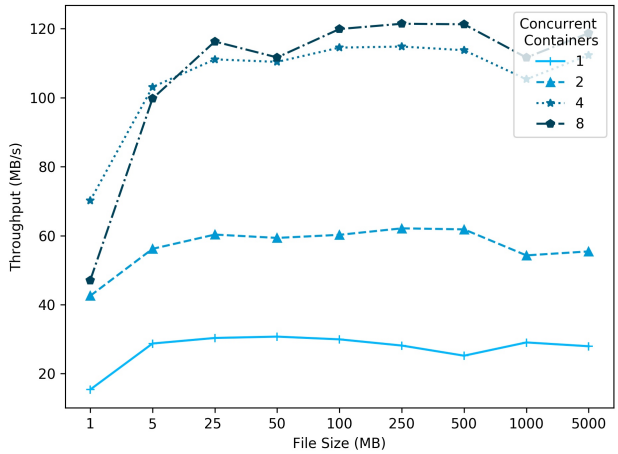
*1) Crawling:* To evaluate the performance of the crawler, we examine crawling latency on CDIAC stored first in an FTP repository, and again with CDIAC downloaded to local disk. Both crawlers run locally, using an Ubuntu 17.04 desktop machine with an Intel Core i7-3770 CPU (3.40GHz) and 16GB DDR3 RAM. The machine is configured with Docker 1.13.1 and launch a single crawler container.

Crawling the FTP server took 34 hours to complete. We then ran the crawler on the local copy using a depth-first ordering. Running the crawler on the same machine as the data took approximately 11 minutes. This means that the crawler itself is generally quite fast, but a nonnegligible amount of latency occurs when we must scan repositories over a network.

*2) Extractor Scalability:* We analyze the performance of an individual extractor to determine how our system performs when processing files of varying size using an increasing number of containers executing concurrently. Our experiments employ multiple hybrid files, each of which start with a short, 10-line preamble followed by a column-header line, with structured data beneath. These files were determined to be a tabular+unstructured hybrid file by the **`sampler`**

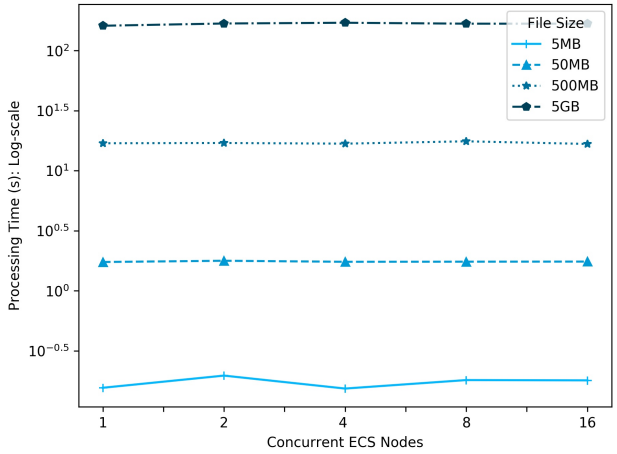



(a) Extraction latency vs. number of container instances, for different file sizes

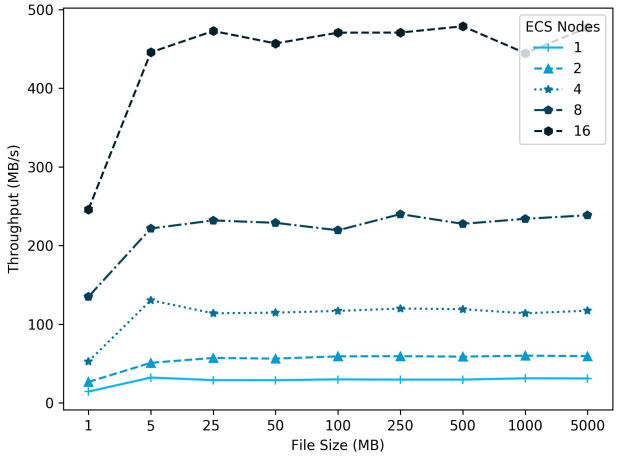



(b) Throughput vs. file size for different numbers of container instances

Fig. 5: Local performance on 1–8 concurrent containers




(a) Extraction latency vs. number of ECS nodes, for different file sizes




(b) Throughput vs. file size, for different number of ECS nodes

Fig. 6: Cloud performance on 1–16 node ECS clusters

extractor, and thus this test uses each of the **`tabular`**, **`topic`**, and **`null`** extractors. Each file size class (5MB–5GB) is a truncated version of a larger 5.2GB data to avoid excessive statistical variance between files.

Figure 5 shows the results when running locally and varying the number of concurrent containers. We see that execution on the local deployment scales linearly as long as a free core is available, but does not scale well beyond this point (in this case, as Skluma upgrades from 4 to 8 concurrent containers).

Cloud deployments were hosted on an ECS cluster, using between 1 and 16 m4.xlarge instances. In this case, we simulate a live Skluma run by populating an S3 object store bucket with the dataset, and use a Simple Queue Service queue to provide a global FIFO file processing ordering. The cloud deployment results, shown in Figure 6, exhibit linear scalability with respect to number of nodes and file size in terms of both processing time and throughput.

*3) Extractor Performance:* Although the results above show Skluma is capable of near-linear scaling when processing hybrid, tabular+unstructured data, the throughput is not necessarily indicative of other extractors. Each Skluma extractor performs a specialized extraction task, where some, such as image extraction, perform much more compute-intensive tasks. Therefore, to forecast Skluma's performance on a given repository, we first quantify the performance of each extractor.

Here we present the average performance of each Skluma extractor when applied to the entire CDIAC repository using the cloud deployment. These results, shown in Table IV, highlight the vastly different resource requirements of extractors, ranging from 3.47 ms for **`null`** to almost 15s for **`map`**.

*4) CDIAC Indexing:* We explore both the wall-time and cloud cost for indexing the entire CDIAC repository on AWS. We use March 2018 AWS spot price data to estimate the costs of running on 1–16-node ECS clusters, accounting for compute time on vanilla m4.xlarge instances in the us-east-1 availability zone; RDS, S3, and SQS costs; and intra-network file transfers. We see in Table V that the entire repository can e processed in less than 5 hours, for just $14.25, when using 16 nodes.

TABLE IV: Performance profiles for the crawler, extractors, and data download from S3 to AWS compute, giving for each the number of CDIAC files processed and, for those files, mean file size and execution time. We separate out decompression task within **`universal`**.

| Pipeline Step | File count | Mean values Size (KB) | Time (ms) |
|---|---|---|---|
| Crawler (local) | 500,001 | 4,734.00 | 0.02 |
| Extractors | | | |
| **`universal`** | 500,001 | 4,734.00 | 388.10 |
| compressed only | 381 | 971.50 | 69.59 |
| **`sampler`** | 500,001 | 4,734.00 | 3.01 |
| **`tabular`** | 173,019 | 180.96 | 21.78 |
| **`null`** | 3,144 | 49.91 | 3.47 |
| **`structured`** | 22,127 | 28.37 | 15.36 |
| **`topic`** | 53,184 | 2,151.00 | 684.50 |
| **`image`** | 32,127 | 117.10 | 6.91 |
| **`map`** | 583 | 201.50 | 14,920.00 |
| Network download | 500,001 | 4,734.00 | 152.20 |

TABLE V: Time and cost to index all CDIAC files on AWS.

| Instances | Time (hours) | Cost ($) |
|---|---|---|
| 1 | 70.30 | 16.61 |
| 2 | 35.10 | 15.35 |
| 4 | 17.60 | 14.72 |
| 8 | 8.80 | 14.40 |
| 16 | 4.49 | 14.25 |

## VI. Related Work

Skluma is not the first system to collect and extract metadata from data. It builds upon a significant prior work related to metadata extraction from images, tabular data, and free text as well as work related to data integration and data lakes.

Dataspaces [26], [27] are an abstraction for providing a structured, relational view over unstructured, unintegrated data. Dataspaces apply a "pay-as-you-go" data integration model in which data integration tasks are performed only when needed and therefore upfront data integration costs are reduced. Like Skluma, dataspaces do not assume complete control over the data, but merely integrate and coordinate the various methods of querying that data. Dataspace research builds upon related work in schema mapping, data integration, managing and querying uncertain data, indexing, and incorporating human curation.

Pioneering research on data lakes has developed methods for extracting standard metadata from nonstandard file types and formats [8]. We ourselves have adapted a data lake model to standardize and extract metadata from strictly geospatial datasets [28]. Normally data lakes are optimized for some sort of institution-specific target, whether transactional, scientific, or networked. Skluma, in contrast, assumes no standard formats, but instead seeks to provide information on related, relevant data and their attributes.

Tools such as RecordBreaker [29] and PADS [10] provide methods for identifying columns; while simplistic, they are surprisingly accurate. The basic approach uses a set of possible delimiters and attempts to parse file rows using this set. For each column, the tools then analyze individual values to determine type. A histogram of values is created. If the values are homogeneous, a basic type can be easily inferred from the histogram. If the values are heterogeneous, then the column is broken into a structure and the individual tokens with the structure are analyzed in a recursive process.

Methods have also been developed for extracting data from web pages (e.g., Google's Web Tables [30]), for example to mine real-estate pages to determine information about properties, such as type, size, number of bedrooms, and location. These tools commonly seek to identify structured components of the page (e.g., using HTML structure) and then apply rules to extract specific information. Systems for extracting information from scientific publications and electronic medical records work in similar ways, by locating structured information and then extracting features. A wide range of techniques have been explored for such tasks, including the use of templates, rules [31], schema mining [32], path expressions [33], and ontologies [34]. Recently many of these techniques have been applied in machine learning approaches [35].

Finally, Skluma supplements work on the cleaning and labeling of data by using context features. Data Civilizer [36] accounts for proximate files in data warehouses by building and interpreting linkage graphs. Others have used context as a means to determine how certain metadata might be leveraged to optimize performance in a specific application or API [37]. Skluma collects and analyzes context metadata in order to allow research scientists to find, query, and download related datasets that aid their scientific efforts.

## VII. Summary

We have described Skluma, a modular, cloud-scalable system for extracting metadata from arbitrary scientific data repositories. We have shown that this system is generalizable through extensive study on diverse file systems across our use cases, scalable via its dynamic allocation of container resources, and extensible such that scientists can build their own extractors with ease.

Future work includes building crawlers for distributed file systems such as Lustre in order to enable Skluma's use in high performance computing environments. We also want to explore crowd-sourcing to evaluate our metadata extractions and to provide feedback in cases where there is insufficient contextual information for machine learning. Additionally, we plan to explore new task-to-container mapping strategies to improve CPU usage and metadata extraction throughput.

## Availability of Locally Deployable Skluma Code

See https://github.com/globus-labs/skluma-local-deploy.


## Acknowledgments

This work was supported in part by the UChicago CERES Center for Unstoppable Computing, DOE contract DE-AC02-06CH11357, and NIH contract 1U54EB020406. We also acknowledge research credits provided by Amazon Web Services and compute hours from Jetstream [38]. We thank Emily Herron, Dr. Shan Lu, Goutham Rajendran, and Chaofeng Wu for their input on extractors.

## REFERENCES


[1] M. Chessell, F. Scheepers, N. Nguyen, R. van Kessel, and R. van der Starre, "Governing and managing big data for analytics and decision makers," 2014, http://www.redbooks.ibm.com/redpapers/pdfs/redp5120.pdf.

[2] K. Chard, J. Pruyne, B. Blaiszik, R. Ananthakrishnan, S. Tuecke, and I. Foster, "Globus data publication as a service: Lowering barriers to reproducible science," in *IEEE 11th International Conference on e-Science*, 2015, pp. 401–410.

[3] J. M. Wozniak, K. Chard, B. Blaiszik, R. Osborn, M. Wilde, and I. Foster, "Big data remote access interfaces for light source science," in *2nd IEEE/ACM International Symposium on Big Data Computing (BDC)*, Dec 2015, pp. 51–60.

[4] M. Egan, S. Price, K. Kraemer, D. Mizuno, S. Carey, C. Wright, C. Engelke, M. Cohen, and M. Gugliotti, "VizieR Online Data Catalog: MSX6C infrared point source catalog. The midcourse space experiment point source catalog version 2.3 (october 2003)," *VizieR Online Data Catalog*, vol. 5114, 2003.

[5] D. Welter, J. MacArthur, J. Morales, T. Burdett, P. Hall, H. Junkins, A. Klemm, P. Flicek, T. Manolio, L. Hindorff *et al.*, "The NHGRI GWAS catalog, a curated resource of SNP-trait associations," *Nucleic Acids Research*, vol. 42, no. D1, pp. D1001–D1006, 2013.

[6] A. Rajasekar, R. Moore, C.-y. Hou, C. A. Lee, R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S.-Y. Chen, L. Gilbert *et al.*, "irods primer: integrated rule-oriented data system," *Synthesis Lectures on Information Concepts, Retrieval, and Services*, vol. 2, no. 1, pp. 1–143, 2010.

[7] G. King, "An introduction to the dataverse network as an infrastructure for data sharing," 2007.

[8] I. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino, "Data wrangling: The challenging journey from the wild to the lake," in *7th Biennial Conference on Innovative Data Systems Research*, 2015.

[9] A. Halevy, F. Korn, N. F. Noy, C. Olston, N. Polyzotis, S. Roy, and S. E. Whang, "Goods: Organizing Google's datasets," in *International Conference on Management of Data*. ACM, 2016, pp. 795–806.

[10] K. Fisher and D. Walker, "The PADS project: An overview," in *14th International Conference on Database Theory*. ACM, 2011, pp. 11–17.

[11] P. Beckman, T. J. Skluzacek, K. Chard, and I. Foster, "Skluma: A statistical learning pipeline for taming unkempt data repositories," in *29th International Conference on Scientific and Statistical Database Management*, 2017, p. 41.

[12] Wikipedia, "List of file formats," "https://en.wikipedia.org/wiki/List_of_file_formats".

[13] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, Mar. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600239.2600241

[14] R. Chard, K. Chard, J. Alt, D. Y. Parkinson, S. Tuecke, and I. Foster, "Ripple: Home automation for research data management," in *IEEE 37th International Conference on Distributed Computing Systems Workshops*. IEEE, 2017, pp. 389–394.

[15] R. Ananthakrishnan, B. Blaiszik, K. Chard, R. Chard, B. McCollam, J. Pruyne, S. Rosen, S. Tuecke, and I. Foster, "Globus platform services for data publication," in *Practice and Experience in Advanced Research Computing (PEARC)*. IEEE, 2018.

[16] N. L. Beebe, L. A. Maddox, L. Liu, and M. Sun, "Sceadan: Using concatenated n-gram vectors for improved file and data type classification," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 9, pp. 1519–1530, Sept 2013.

[17] R. B. Tchoua, K. Chard, D. Audus, J. Qin, J. de Pablo, and I. Foster, "A hybrid human-computer approach to the extraction of scientific facts from the literature," *Procedia Computer Science*, vol. 80, pp. 386–397, 2016.

[21] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[18] "Geograpy," https://github.com/ushahidi/geograpy.

[19] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1532–1543.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[22] R. Smith, "An overview of the Tesseract OCR engine," in *9th International Conference on Document Analysis and Recognition*, vol. 2. IEEE, 2007, pp. 629–633.

[23] S. Biswas and A. K. Das, "Text extraction from scanned land map images," in *International Conference on Informatics, Electronics & Vision*. IEEE, 2012, pp. 231–236.

[24] S. Suzuki *et al.*, "Topological structural analysis of digitized binary images by border following," *Computer vision, graphics, and image processing*, vol. 30, no. 1, pp. 32–46, 1985.

[25] "Natural Earth Public Domain Map Datasets," http://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-admin-0-details/.

[26] M. Franklin, A. Halevy, and D. Maier, "From databases to dataspaces: a new abstraction for information management," *ACM Sigmod Record*, vol. 34, no. 4, pp. 27–33, 2005.

[27] A. D. Sarma, X. L. Dong, and A. Y. Halevy, "Data modeling in dataspace support platforms," in *Conceptual Modeling: Foundations and Applications*. Springer, 2009, pp. 122–138.

[28] T. J. Skluzacek, K. Chard, and I. Foster, "Klimatic: A virtual data lake for harvesting and distribution of geospatial data," in *1st Joint International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems*, 2016, pp. 31–36.

[29] Cloudera, "Recordbreaker," 2014, https://github.com/cloudera/RecordBr-eaker/tree/master/src. Visited Feb. 28, 2017.

[30] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "WebTables: Exploring the power of tables on the web," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 538–549, 2008.

[31] M. J. Cafarella, A. Halevy, and J. Madhavan, "Structured data on the web," *Communications of the ACM*, vol. 54, no. 2, pp. 72–79, 2011.

[32] L. Singh, B. Chen, R. Haight, P. Scheuermann, and K. Aoki, "A robust system architecture for mining semi-structured data." in *KDD*, 1998, pp. 329–333.

[33] C. Enhong and W. Xufa, "Semi-structured data extraction and schema knowledge mining," in *25th EUROMICRO Conference*, vol. 2, 1999, pp. 310–317 vol.2.

[34] K. Taniguchi, H. Sakamoto, H. arimura, S. Shimozono, and S. Arikawa, "Mining semi-structured data by path expressions," pp. 378–388, 2001.

[35] K.-W. Tan, H. Han, and R. Elmasri, "Web data cleansing and preparation for ontology extraction using WordNet," in *1st International Conference on Web Information Systems Engineering*, vol. 2, 2000, pp. 11–18 vol.2.

[36] D. Deng, R. C. Fernandez, Z. Abedjan, S. Wang, M. Stonebraker, A. Elmagarmid, I. F. Ilyasl, S. Madden, M. Ouzzani, and N. Tang, "The data civilizer system," in *8th Biennial Conference on Innovative Data Systems Research*, 2017.

[37] J. M. Hellerstein, V. Sreekanti, J. E. Gonzalez, J. Dalton, A. Dey, S. Nag, K. Ramachandran, S. Arora, A. Bhattacharyya, S. Das *et al.*, "Ground: A data context service," in *8th Biennial Conference on Innovative Data Systems Research*, 2017.

[38] C. A. Stewart, T. M. Cockerill, I. Foster, D. Hancock, N. Merchant, E. Skidmore, D. Stanzione, J. Taylor, S. Tuecke, G. Turner, M. Vaughn, and N. I. Gaffney, "Jetstream: A self-provisioned, scalable science and engineering cloud environment," in *XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*. ACM, 2015, pp. 29:1–29:8. [Online]. Available: http://doi.acm.org/10.1145/2792745.2792774