Delta: An Open Source Data Service for Research

Lexington Whalen
Department of Computer Science
University of South Carolina
Columbia, United States
LAWHALEN@email.sc.edu

Homayoun Valafar
Department of Computer Science
University of South Carolina
Columbia, United States
homayoun@cec.sc.edu

***Abstract*—Every research project necessitates data, often requiring sharing and collaborative review within a team. Traditionally, services like Dropbox [2], Google Drive [1], or OneDrive [3] have been employed for storing research data. However, the limitations of these platforms become evident with the growing scale of collected data. Existing file-sharing services generally mandate paid subscriptions for increased storage or additional members, diverting research funds from addressing the core research problem that a lab is attempting to work on. Moreover, these services often lack direct features for reviewing or commenting on data quality, a vital part of ensuring high quality data generation. In response to these challenges, we present Delta, a specialized file transfer service crafted for specifically for researchers. Delta operates as an application hosted on a research lab server. This design ensures that, with access to a machine and an internet connection, teams can facilitate file storage, transfer, and review without incurring extra costs. Being an open-source project, Delta can be customized to suit the unique requirements of any research team, and is able to evolve to meet the needs of the research community. We open source the code here: https://github.com/lxaw/Delta.**

***Index Terms*—data transfer, data storage, data reviewing, data sharing**

I. Introduction

There is a noticeable dearth of quality open source research services with regards to data. Most data services are merely file sharing ones, that only allow file upload, download, and organization. While this does allow researchers to transmit their collected data among another, it lacks many critical features that most researchers would like to have, such as quality control, commenting, tagging, organization under groups, annotation, and more. Furthermore, these services are oftentimes prohibitively expensive to smaller research labs, and their free versions have limitations on both number of users and amount of storage. The current state of research is to use services such as Microsoft's OneDrive [3], Dropbox [2], or Google Drive [1]. Each of these services have limitations in terms of storage capacity, user management, and feature set that make them suboptimal for research data management [17].

Furthermore, the modern times have seen unprecedented growth in the amount of data collected. The era of big data has brought about massive datasets, particularly in fields like genomics, astronomy, and social media analytics [14]. Machine learning and deep learning have further amplified the need for large, high-quality datasets for training models [8]. The Internet of Things (IoT) is another significant contributor to the data deluge, with billions of connected devices generating continuous streams of data [9]. Even at a personal level, the proliferation of smartphones and digital services has led to an explosion of user-generated data [10].

This rapid growth in data volume and variety has outpaced the capabilities of traditional data management tools and practices. Researchers often struggle with the challenges of storing, organizing, sharing, and collaborating on large datasets [11]. The lack of specialized tools for research data management leads to ad-hoc solutions, data silos, and inefficiencies in the research process [12].

To address these challenges, we present Delta, an open source data service designed specifically for the needs of researchers. Delta provides a platform for efficient data storage, sharing, and collaboration, with features tailored to the research workflow. By leveraging open source technologies and a modular architecture, Delta enables customization and extensibility to meet the diverse requirements of different research domains. The open source nature of Delta also promotes transparency, reproducibility, and community-driven development [13].

II. The Problem

In the era of data-driven research, scientists across various disciplines are grappling with an unprecedented influx of data. From high-throughput sequencing in genomics [14] to large-scale simulations in physics [15], researchers are generating and collecting massive datasets at an astounding rate. However, managing this data effectively poses significant challenges that can hinder research productivity and collaboration.

One of the primary issues researchers face is the need for efficient data labeling and annotation. Raw data often requires contextualization and metadata to be meaningful and usable for analysis [12]. Manually annotating large datasets is time-consuming and prone to inconsistencies, especially when multiple researchers are involved. The lack of standardized annotation frameworks and tools further compounds this problem, leading to a fragmented and inefficient data labeling process.

Quality control is another critical concern in research data management. Ensuring the accuracy, completeness, and reliability of data is essential for drawing valid conclusions and reproducing results [16]. However, reviewing and validating large datasets manually is a daunting task, particularly in collaborative projects where data is collected by multiple individuals or teams. The absence of systematic quality control mechanisms can lead to errors, inconsistencies, and a lack of trust in the data.

Effective organization and management of research data is also a significant challenge, both at the individual researcher level and within research groups. Principal Investigators (PIs) need to keep track of various datasets, their versions, and associated metadata, often relying on ad-hoc solutions like spreadsheets or file naming conventions [17]. At the group level, data silos can emerge when different teams use incompatible data formats or storage systems, hindering collaboration and data integration.

Sharing data between research groups is another pain point in the current research landscape. Collaborative projects often involve multiple institutions and disciplines, each with their own data management practices and infrastructure [18]. Incompatible data formats, lack of standardization, and concerns about data ownership and attribution can impede smooth data sharing and reuse. Moreover, transferring large datasets across institutional boundaries can be time-consuming and subject to security and privacy constraints.

Addressing these challenges requires a comprehensive and integrated approach to research data management. Researchers need tools and platforms that streamline data annotation, enable robust quality control, facilitate organization and discovery, and promote seamless data sharing and collaboration. Existing solutions, such as generic cloud storage services or ad-hoc scripts, often fall short in terms of functionality, scalability, and ease of use. There is a clear need for a purpose-built, open-source data management platform that caters to the unique requirements of scientific research.

III. Current Alternatives

Several cloud storage providers offer APIs and developer tools that enable programmatic access to their services. For example:

- Google Drive provides a REST API [1] for uploading, downloading, searching, and manipulating files stored on Google Drive. Client libraries are available in multiple programming languages.
- Dropbox offers a similar REST API [2] with SDKs for major platforms to integrate Dropbox capabilities into applications. Key features include file upload/download, sharing, search, and user management.
- Microsoft OneDrive also has a comprehensive REST API [3] for accessing OneDrive files, folders, and other data. Client libraries support integration with web, mobile, and desktop apps.
- Box provides a content management platform with a REST API [4] for building custom applications. It offers features like file preview, version control, and granular access permissions, catering to enterprise needs.
- Amazon S3 (Simple Storage Service) is a scalable object storage service with an API [5] for storing and retrieving data. While not primarily designed for end-user file management, it's often used as a foundation for building cloud storage applications.

While these APIs enable building custom applications with cloud storage, the underlying limitations of the services still apply - costs scale with storage and user needs, and specialized features for research data management are lacking. Leveraging the APIs still requires significant development effort to create a tailored solution. Furthermore, some services like Amazon S3 are more suited to developers and lack user-friendly interfaces out-of-the-box. Furthermore, these services are not open sourced, so any modifications (i.e. addition of service, design change, bug fix) cannot be done directly by the users. We make such issues more clear in *Issues with Current Approaches*.

IV. Issues with Current Approaches

While current cloud storage services offer APIs and enable building custom applications, they have several notable shortcomings, especially in the context of research data management:

1) **Cost:** Services like Google Drive, Dropbox, and OneDrive can become expensive as data storage needs grow, often requiring paid subscriptions for additional storage or users. This diverts funds from core research activities.
2) **Lack of Specialization:** General-purpose storage services lack features tailored for research, such as data quality control, peer review workflows, metadata management, and data provenance tracking.
3) **Limited Customization:** Although APIs enable custom app development, the underlying platforms cannot be easily modified or extended. Researchers cannot add new features or modify existing behavior to suit their specific needs.
4) **Vendor Lock-In:** By relying on proprietary services, researchers risk being locked into a particular vendor's ecosystem. Migration to alternative platforms can be difficult and costly.
5) **Data Ownership and Control:** With commercial services, there may be ambiguity around data ownership and control. Researchers may have concerns about intellectual property rights and the ability to access their data if a service is discontinued.
6) **Data Privacy and Security:** Storing sensitive research data on third-party servers raises privacy and security concerns. Researchers may be hesitant to entrust confidential data to external providers.
7) **Collaboration Barriers:** While cloud services facilitate file sharing, they often lack advanced collaboration features like real-time co-authoring, version control, and granular access controls that are vital for research teams.
8) **Integration Challenges:** Integrating cloud storage with existing research tools and workflows can be challenging. Researchers may need to develop custom glue code or rely on limited third-party integrations.
9) **Dependency on Internet Connectivity:** Cloud services require reliable internet access, which can be a constraint

in field research settings or areas with limited connectivity.

10) **Long-Term Preservation:** Commercial services may not prioritize long-term data preservation, which is crucial for research reproducibility and data archiving. There may be uncertainties about data durability and accessibility over extended periods.

An open source solution like Delta addresses these issues by providing a specialized, customizable, and cost-effective platform for research data management. By hosting Delta on their own infrastructure, research teams have full control over their data, can tailor the platform to their specific needs, and avoid vendor lock-in. The open source nature ensures transparency, enables community-driven development, and allows for integration with a wide range of tools. Moreover, an open source solution can be deployed in local or offline environments, mitigating concerns about internet connectivity and data privacy. Delta empowers researchers to manage their data on their own terms, prioritizing the unique requirements of scientific research.

V. Delta Architecture

Delta is built using a modern web stack consisting of Django [25], Django REST Framework [26], React [30], and Redux [32]. This architecture was chosen to enable fast, scalable development, cross-platform compatibility, and a smooth user experience.

A. Backend

On the backend, Delta utilizes the Django web framework. Django is a high-level Python framework that encourages rapid development and clean, pragmatic design [25]. It is known for its robustness, scalability, and extensive community support, making it an ideal choice for building the backend of Delta.

To facilitate communication between the frontend and backend, Delta employs the Django REST Framework (DRF). DRF is a powerful and flexible toolkit for building Web APIs [26]. A REST (Representational State Transfer) API is an architectural style for designing networked applications. It defines a set of constraints and properties based on HTTP, such as statelessness, client-server separation, and a uniform interface. RESTful APIs use HTTP methods (GET, POST, PUT, DELETE) to perform CRUD (Create, Read, Update, Delete) operations on resources, which are identified by URLs.

DRF allows for easy creation of API endpoints that can be accessed from various platforms and devices. It provides a set of tools and abstractions for building RESTful APIs quickly and efficiently. DRF's serialization system enables seamless conversion of Django models to various content types, such as JSON or XML, making it ideal for building APIs that can be consumed by different clients.

One of the key benefits of using a RESTful API is that it provides a clear and consistent interface for interacting with the backend. It allows for a separation of concerns between the client and the server, enabling independent development and scaling of the frontend and backend. RESTful APIs also promote statelessness, meaning that each request contains all the necessary information to be processed, making the application more scalable and easier to cache.

For authentication, Delta uses Knox tokens, which provide enhanced security compared to the default TokenAuthentication in DRF. Knox tokens are hashed using SHA-512 and encrypted in the database, making them more secure in case of a database compromise. Additionally, Knox tokens have a configurable expiry time, adding an extra layer of security. Token-based authentication is stateless, meaning that the server does not need to keep a record of which users are logged in. This makes the authentication process more scalable and reduces the load on the server.

Delta utilizes a SQLite database by default, but this can be easily modified to use other popular databases such as MongoDB [27], PostgreSQL [28], or MySQL [29]. Django's ORM (Object-Relational Mapping) allows for easy interaction with the database through the use of models. A model is a Python class that subclasses django.db.models.Model and represents a single database table. Each attribute of the model represents a database field. This abstraction provides a convenient and intuitive way to define the structure and behavior of the data being stored, while Django handles the underlying database operations.

B. Frontend

For the frontend, Delta uses React, a popular JavaScript library for building user interfaces [30]. React's component-based architecture allows for the creation of reusable UI components, which can be easily composed to build complex user interfaces. This modular approach makes the codebase more maintainable and easier to understand. React's virtual DOM (Document Object Model) implementation is another key feature that contributes to its efficiency. Instead of directly manipulating the browser's DOM, React maintains a lightweight copy of the DOM in memory called the virtual DOM. When data changes, React updates the virtual DOM first, calculates the differences (diff) between the virtual and real DOM, and then efficiently updates the browser's DOM with minimal changes. This approach significantly improves performance, especially in large-scale applications with frequent data updates.

React's declarative syntax, JSX (JavaScript XML), allows developers to write HTML-like code within JavaScript. This makes the code more readable and intuitive, as the structure of the UI closely resembles the final rendered output. React's one-way data flow, also known as unidirectional data flow, ensures that data flows in a single direction from parent components to child components. This makes the application's state more predictable and easier to debug.

To manage the application state, Delta incorporates Redux, a predictable state container for JavaScript apps [32]. Redux follows the principles of a single source of truth, state immutability, and pure functions for state modifications. It maintains a single, immutable state tree (store) that represents the entire application state. Components can access the required parts of the state by subscribing to the store. To modify the state, components dispatch actions, which are plain JavaScript objects describing the desired changes. Reducers, pure functions that take the current state and an action as input and return a new state, handle these actions and update the state accordingly. This centralized state management makes it easier to reason about the application's behavior, enables features like undo/redo, and facilitates state persistence.

The combination of React and Redux allows for a highly responsive and interactive user interface. When a user interacts with the application, such as creating a new post or rating a dataset, React components dispatch the corresponding actions. Redux processes these actions, updates the state, and notifies the subscribed components. React then efficiently re-renders the affected components, updating the UI in real-time without requiring a full page refresh. This provides a smooth and engaging user experience, similar to that of modern social media platforms.

To communicate with the backend API, Delta utilizes the Axios library [33]. Axios is a popular, promise-based HTTP client for JavaScript that simplifies making API requests. It provides an easy-to-use API for sending GET, POST, PUT, and DELETE requests to the server and handles the response data efficiently. Axios also supports request and response interceptors, which allow for easy modification or examination of requests and responses before they are sent or received. This is particularly useful for adding authentication headers, handling errors, or transforming data.

When a user interacts with the application, React components dispatch actions that trigger API requests using Axios. Axios sends the requests to the appropriate backend endpoints, along with any necessary data or parameters. The backend processes these requests, interacts with the database, and returns the response data in JSON format. Axios then resolves the promise, and the received data is dispatched to the Redux store. The store updates the state, and the subscribed React components re-render with the new data, updating the UI accordingly. This seamless integration between React, Redux, and Axios enables a smooth and efficient data flow between the frontend and backend, resulting in a responsive and dynamic user experience.

C. Integration

The Django backend and React frontend are integrated through REST API calls, enabling seamless communication and data exchange between the two components. When a user interacts with the frontend, such as submitting a form or clicking a button, React components dispatch actions that trigger API requests to the Django backend. These actions are typically handled by Redux middleware, such as Redux Thunk [34] or Redux Saga [35], which allows for asynchronous operations and side effects.

The API requests are sent using the Axios library, which is configured with the appropriate backend API endpoints. Axios sends HTTP requests (GET, POST, PUT, DELETE) to the corresponding Django views, along with any necessary data or parameters. The Django views, defined in the views.py files, handle these requests and perform the required processing. They interact with the database using Django's ORM (Object-Relational Mapping) and execute any necessary business logic.

After processing the request, the Django views return the appropriate JSON responses using Django REST Framework's serializers. Serializers convert Django model instances or querysets into JSON format, which can be easily consumed by the frontend. The JSON responses typically include the requested data, such as a list of objects, a single object, or a success/error status.

Axios receives the JSON response from the backend and resolves the promise. The received data is then dispatched to the Redux store using the appropriate action creators. The Redux reducers handle these actions and update the application state accordingly. React components that are subscribed to the relevant parts of the state are automatically re-rendered with the new data, updating the UI in real-time.

This architecture allows for a clear separation of concerns between the frontend and backend. The backend focuses on data processing, business logic, and database management, while the frontend handles user interactions and UI rendering. This decoupling makes the application more maintainable and allows for independent development and scaling of the frontend and backend.

The use of REST APIs provides a standardized and platform-agnostic interface for communication between the frontend and backend. This allows for flexibility in terms of the technologies used on either end. For example, the frontend could be implemented using a different framework or library, such as Angular [36] or Vue.js [37], while still being able to communicate with the Django backend through the same API endpoints. Similarly, the backend could be swapped with a different framework or language, such as Flask [38] or Node.js [39], as long as it adheres to the same API contract.

By leveraging the power of Django, Django REST Framework, React, Redux, and Axios, Delta provides a robust, scalable, and user-friendly platform for researchers and data scientists to manage, share, and collaborate on their data. The integration of these technologies enables a seamless and efficient data flow between the frontend and backend, resulting in a responsive and interactive user experience. The modular architecture and clear separation of concerns facilitate maintainability, extensibility, and independent scaling of the application components.

VI. Current Product

[write]

VII. Future Goals

[write]

References

[1] Google, "Google Drive: Free Cloud Storage for Personal Use," Google.com, 2019. https://www.google.com/drive/

[2] Dropbox, "Dropbox," Dropbox, 2018. https://www.dropbox.com/

[3] “Personal Cloud Storage – Microsoft OneDrive,” Microsoft.com, 2024. https://www.microsoft.com/en-us/microsoft-365/onedrive/

[4] Box, ”Box Developer Documentation,” Box, 2023. https://developer.box.com/

[5] Amazon Web Services, ”Amazon S3 API Reference,” Amazon.com, 2023. https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html

[6] D. R. Kowalczyk and S. Y. Shankar, ”Data sharing in the sciences,” *Annual Review of Information Science and Technology*, vol. 45, no. 1, pp. 247-294, 2011, doi: 10.1002/aris.2011.1440450113.

[7] Z. D. Stephens et al., ”Big Data: Astronomical or Genomical?” *PLOS Biology*, vol. 13, no. 7, p. e1002195, Jul. 2015, doi: 10.1371/journal.pbio.1002195.

[8] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, ”Revisiting Unreasonable Effectiveness of Data in Deep Learning Era,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 843-852, doi: 10.1109/ICCV.2017.97.

[9] M. A. Khan, K. Salah, ”IoT security: Review, blockchain solutions, and open challenges,” *Future Generation Computer Systems*, vol. 82, pp. 395-411, 2018, doi: 10.1016/j.future.2017.11.022.

[10] D. Reinsel, J. Gantz, and J. Rydning, ”The Digitization of the World - From Edge to Core,” *IDC White Paper*, Nov. 2018. [Online]. Available: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf

[11] J. C. Wallis, E. Rolando, and C. L. Borgman, ”If We Share Data, Will Anyone Use Them? Data Sharing and Reuse in the Long Tail of Science and Technology,” *PLOS ONE*, vol. 8, no. 7, p. e67332, Jul. 2013, doi: 10.1371/journal.pone.0067332.

[12] M. D. Wilkinson et al., ”The FAIR Guiding Principles for scientific data management and stewardship,” *Scientific Data*, vol. 3, no. 1, Art. no. 1, Mar. 2016, doi: 10.1038/sdata.2016.18.

[13] J. D. Gezelter, ”Open Source and Open Data Should Be Standard Practices,” *The Journal of Physical Chemistry Letters*, vol. 6, no. 7, pp. 1168-1169, Apr. 2015, doi: 10.1021/acs.jpclett.5b00285.

[14] Z. D. Stephens et al., ”Big Data: Astronomical or Genomical?” *PLOS Biology*, vol. 13, no. 7, p. e1002195, Jul. 2015, doi: 10.1371/journal.pbio.1002195.

[15] M. Vogelsberger et al., ”Introducing the Illustris Project: Simulating the coevolution of dark and visible matter in the Universe,” *Monthly Notices of the Royal Astronomical Society*, vol. 444, no. 2, pp. 1518-1547, Oct. 2014, doi: 10.1093/mnras/stu1536.

[16] D. A. Eisner, ”Reproducibility of science: Fraud, impact factors and carelessness,” *Journal of Molecular and Cellular Cardiology*, vol. 114, pp. 364-368, Jan. 2018, doi: 10.1016/j.yjmcc.2017.10.009.

[17] D. R. Kowalczyk and S. Y. Shankar, ”Data sharing in the sciences,” *Annual Review of Information Science and Technology*, vol. 45, no. 1, pp. 247-294, 2011, doi: 10.1002/aris.2011.1440450113.

[18] C. Tenopir et al., ”Data Sharing by Scientists: Practices and Perceptions,” *PLOS ONE*, vol. 6, no. 6, p. e21101, Jun. 2011, doi: 10.1371/journal.pone.0021101.

[19] Django Software Foundation, ”Django Documentation,” Django Project, 2023. [Online]. Available: https://docs.djangoproject.com/

[20] Django REST Framework, ”Django REST Framework,” Django REST Framework, 2023. [Online]. Available: https://www.django-rest-framework.org/

[21] React, ”React – A JavaScript library for building user interfaces,” React, 2023. [Online]. Available: https://reactjs.org/

[22] Redux, ”Redux - A Predictable State Container for JS Apps,” Redux, 2023. [Online]. Available: https://redux.js.org/

[23] Django Software Foundation, ”Sites Using Django,” Django Project, 2023. [Online]. Available: https://djangosites.org/

[24] J. Neuhaus, ”16 Popular Websites Built With ReactJS,” Honeypot, 2022. [Online]. Available: https://www.honeypot.io/blog/popular-websites-built-with-reactjs/

[25] Django Software Foundation, ”Django Documentation,” Django Project, 2023. [Online]. Available: https://docs.djangoproject.com/

[26] Django REST Framework, ”Django REST Framework,” Django REST Framework, 2023. [Online]. Available: https://www.django-rest-framework.org/

[27] MongoDB, ”MongoDB Documentation,” MongoDB, 2023. [Online]. Available: https://docs.mongodb.com/

[28] PostgreSQL, ”PostgreSQL Documentation,” PostgreSQL, 2023. [Online]. Available: https://www.postgresql.org/docs/

[29] MySQL, ”MySQL Documentation,” MySQL, 2023. [Online]. Available: https://dev.mysql.com/doc/

[30] React, ”React – A JavaScript library for building user interfaces,” React, 2023. [Online]. Available: https://reactjs.org/

[31] J. Neuhaus, ”16 Popular Websites Built With ReactJS,” Honeypot, 2022. [Online]. Available: https://www.honeypot.io/blog/popular-websites-built-with-reactjs/

[32] Redux, ”Redux - A Predictable State Container for JS Apps,” Redux, 2023. [Online]. Available: https://redux.js.org/

[33] Axios, ”Axios - Promise based HTTP client for the browser and Node.js,” Axios, 2023. [Online]. Available: https://axios-http.com/

[34] Redux Thunk, ”Redux Thunk - Thunk middleware for Redux,” Redux Thunk, 2023. [Online]. Available: https://github.com/reduxjs/redux-thunk

[35] Redux Saga, ”Redux Saga - An alternative side effect model for Redux apps,” Redux Saga, 2023. [Online]. Available: https://redux-saga.js.org/

[36] Angular, ”Angular - One framework. Mobile & desktop.” Angular, 2023. [Online]. Available: https://angular.io/

[37] Vue.js, ”Vue.js - The Progressive JavaScript Framework,” Vue.js, 2023. [Online]. Available: https://vuejs.org/

[38] Flask, ”Flask - Web development, one drop at a time,” Flask, 2023. [Online]. Available: https://flask.palletsprojects.com/

[39] Node.js, ”Node.js - About,” Node.js, 2023. [Online]. Available: https://nodejs.org/en/about/