2024 International Conference on Computing, Networking and Communications (ICNC): Next Generation Networks and Internet Applications

Hydra: A Scalable Decentralized P2P Storage Federation for Large Scientific Datasets

Justin Presley**, Xi Wang**, Xusheng Ai†, Tianyuan Yu§, Tym Brandel*, Proyash Podder‡,
Varun Patil§, Alex Afanasyev‡, F. Alex Feltus†, Lixia Zhang§, Susmit Shannigrahi*
*Tennessee Tech
†Clemson University
‡Florida International University
§UCLA

***Abstract*—An increasingly collaborative and distributed nature of scientific collaborations, along with the exploding volume and variety of datasets point to an urgent need for data publication frameworks that allow researchers to publish data rapidly and reliably. However, current scientific data publication solutions only support any one of these requirements at a time. Currently, the most common data publication models are either centralized or ad-hoc. While the centralized model (e.g., publishing via a repository controlled by a central organization) can provide reliability through replication, the publication speed tends to be slower due to the inevitable curation and processing delays. Further, such centralized models may place restrictions regarding what data can be published through them. On the contrary, ad-hoc models lead to concerns such as the lack of replication and a robust security model.**

We present *Hydra*, a peer-to-peer, decentralized storage system that enables decentralized and reliable data publication capabilities. Hydra enables collaborating organizations to create a loosely interconnected and federated storage overlay atop community provided storage servers. The Hydra overlay is entirely decentralized. Hydra enables secure publication and access to data from anywhere and ensures automatic replication of published data, enhancing availability and reliability. Hydra also makes replication decisions without a central controller while accommodating local policies. Hydra embodies a significant stride toward next-generation scientific data management, fostering a decentralized, reliable, and accessible system that fits the changing landscape of scientific collaborations.

I. INTRODUCTION

Several scientific communities, such as genomics, climate science, and high-energy particle physics, are increasingly focusing on data-driven science. Advancements in computational capabilities, higher-resolution sensing devices, and state-of-the-art data processing facilities collectively contribute to an exponential increase in the volume and diversity of generated datasets. Concurrently, scientific research architecture is transforming from a predominantly centralized model to a more globalized and decentralized paradigm. Within this evolving context, scientific collaborations are becoming increasingly ad-hoc and peer-to-peer in nature [2].

Unfortunately, the current data publication and access models do not match this evolving paradigm. In the current scientific landscape, datasets are often distributed through centralized repositories, limiting the pace of data publication and imposing restrictions on the types and volumes of datasets that can be made publicly available. Such a centralized model introduces several additional technical challenges [4]. A centralized controller in this model not only serves as a single point of failure, compromising system resilience and data integrity, but also lacks adaptability to network partitions or congestion, disrupting operational continuity. Furthermore, this centralized approach consolidates all security and access control responsibilities at the central controller, introducing potential bottlenecks and heightened security risks.

These restrictions often lead researchers to use ad-hoc repositories. However, these repositories lack critical features like automated data replication, fault tolerance, standard publication and access APIs, and robust search capabilities. This limitation hinders data reliability and accessibility, contributing to the overall data management problem in scientific research.

To address these challenges, we introduce Hydra, a software framework for building decentralized, peer-to-peer storage federations using community-provided servers. Hydra establishes this federation and eliminates the need for a central controller by leveraging the primitives of Named Data Networking (NDN) [12]. Hydra enhances resiliency through automated data replication while providing individual nodes with complete control over what types of data they store. This control, exercised using the *Favor* parameter, guide data replication decisions. Hydra also integrates automated recovery from node failures. Hydra establishes a robust data federation that improves data publication, access, and compliance with the FAIR [9] principles – Findability, Accessibility, Interoperability, and Reusability.

The primary contributions of this work are as follows: (a) we describe the building blocks and system architecture needed to build a secure decentralized storage federation; (b) we discuss our experiences in a proof-of-concept implementation, preliminary deployment, and evaluations; (c) we discuss how the system benefits science workflows in the context of data publication, access, and FAIR principles.

II. SYSTEM OVERVIEW

The Hydra framework solves the problems of data publication and reliable access by creating a federation of geographically distributed data repositories directly connected. The high-level data and control flow of Hydra are illustrated in Fig. 1. Hydra's architectural blueprint comprises two principal components:

1) **Storage nodes** Hydra stores data on a set of storage nodes provided by members of the federation, which may

---

**These authors contributed equally to this manuscript

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

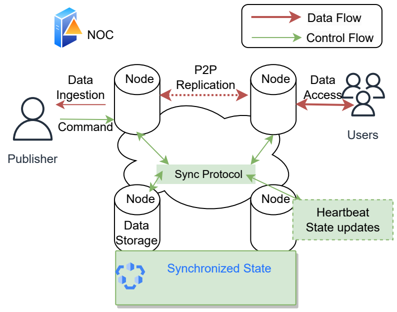

Fig. 1: Hydra's system overview

be organizations or individual researchers. These nodes form a peer-to-peer federation and handle all functions of data publication, access, and replication. At the same time, operators of individual nodes retain autonomy over the nodes they control; they may define storage-level policies and exercise control over the datasets replicated to their nodes. Nodes may also freely join and leave the federation at any time.

2) **Network Operations Center** The Hydra NOC assumes the critical role of disseminating certificates, thus establishing and maintaining trust relationships among the nodes and users in the federation. Note that the NOC's role is limited to certificate distribution, and it does not exercise any other control over the Hydra infrastructure. The NOC can also be replicated for resiliency. As a result, the NOC is not a single point of failure in a running Hydra federation.

The Hydra federation operates as a fully distributed system. Achieving consensus among multiple nodes in such a system can be computationally expensive. Hydra mitigates this challenge by maintaining a synchronized state across nodes using an existing publish-subscribe framework implemented over NDN [6]. This approach eliminates the need for consensus-building during data insertion and replication.

The distinguishing feature of Hydra is the usage of a semantic name on each piece of data. Through the utilization of name-based APIs and anycast routing via Named Data Networking (NDN), Hydra enables the "publication and access from anywhere" of data. Hydra eliminates the conventional methodology of providing end-user data locations. Instead, it uses names as the primary identifier for objects stored within the system. These names are used for all object-related operations, including publication, access, replication, and security and data validation.

In summary, the design of the Hydra framework inherently supports decentralization in multiple ways [3]. First, the Network Operations Center (NOC) focuses exclusively on the dissemination of certificates, establishing trust without exercising additional control over the infrastructure. Second, the peer-to-peer federation of storage nodes under the Hydra framework offers member organizations complete operational autonomy, allowing them to set their own storage-level policies. Finally, Hydra leverages Named Data Networking (NDN) and name-based APIs to eliminate the need for data locations, further advancing its decentralized architecture.

III. ARCHITECTURAL PIECES

This section discusses the architectural pieces needed to create this peer-to-peer storage server federation. Fig. 2 shows these building blocks. It also shows the necessary interactions between these pieces that we describe later in Section IV.

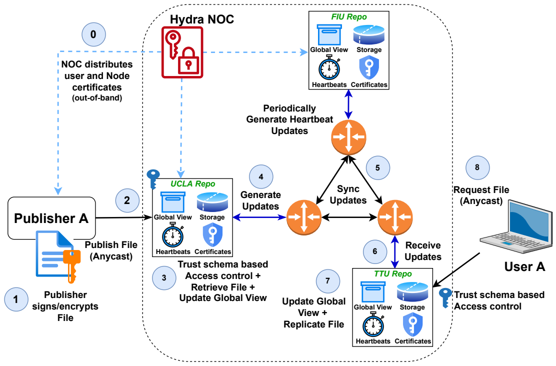

Fig. 2: A high-level overview of Hydra [5]

A. *Base Communication Infrastructure*

Underlying Infrastructure: The Hydra federation comprises storage endpoints, known as "Hydra nodes." These nodes must have network connectivity among themselves, which can be either through TCP/IP over Layer 3 or directly atop Ethernet at Layer 2. Hydra employs Named Data Networking (NDN) as the transport layer on these underlying connectivity layers. When nodes use TCP/IP, NDN operates as an overlay. If nodes connect at Layer 2, such as over VLANs, the NDN transport protocol runs directly over this underlay without needing TCP/IP. We have based these networking configurations on well-documented methods already used in both local and wide-area NDN testbeds [11], so we do not repeat those steps here. To participate in this federation, each node must establish a Hydra endpoint by installing the requisite Hydra software and subsequently joining an NDN-based publish-subscribe group [6] among the nodes.

NDN-based anycast: NDN's anycast functionality plays a pivotal role in directing both data publication and consumption requests to appropriate nodes in a Hydra federation. Anycast in NDN is particularly effective because it enables a single name or identifier to map multiple endpoints natively. When a data publisher or consumer initiates a request, the NDN anycast mechanism assesses the network topology and routes the request to the most suitable or nearest node in the Hydra federation. This is particularly beneficial for data publishers because it allows them to upload their data to the closest or most efficient storage node, optimizing resource utilization and reducing latency. Similarly, data consumers can retrieve data from the most convenient location, which could be determined by proximity, load, or even the cost associated with data retrieval.

Forwarding hints In NDN, forwarding hints are additional information attached to Interest packets, guiding them through specified forwarding paths, potentially bypassing the default routing protocols. In the context of Hydra, forwarding hints are used to redirect a request to a specific node when a contacted node does not have a particular piece of data.

B. Security Primitives

This section describes the security building blocks necessary for Hydra's function.

Node Bootstrapping: In the Hydra ecosystem, each participating node undertakes a security bootstrapping process to acquire essential components for subsequent secure communications, including a trust anchor, NDN certificate, and trust schema, so that Hydra nodes can sign and validate NDN packets securely. The node bootstrapping process includes achieving mutual authentication between NOC and new node, installing trust anchor and certificate from Hydra software package, getting name assignment, and request certificate from the NOC using the NDNCERT protocol [13]. Provided that the NOC successfully validates the new node's credentials and verifies the node's eligibility to join the federation, a certificate is issued. The node uses this certificate to sign all subsequent messages.

Publisher Bootstrapping: In the Hydra ecosystem, publishers – those who can manipulate files – undergo a security bootstrapping process. Distinct from node bootstrapping, user trust establishment has some specificities. First, an email address serves as a publisher's verifiable identifier, authenticated through OAuth mechanisms like campus-based or Google accounts. The NOC keeps an internal database mapping email address authorized to publish datasets. This database relies on pre-existing real-world relationships, such as those between Principal Investigators (PIs) and their students. Second, publishers can only modify namespaces for which they possess the requisite certificates. For example, a publisher who publishes data under the namespace "`/human/genome/dna/hg38`" could perform file operations under that specific namespace. After this initial setup, the remaining bootstrapping process becomes automated. When a publisher requests a certificate for a particular namespace, the NOC cross-references its database to validate the request. A certificate is issued upon successful verification, enabling the publisher to conduct secure operations within the Hydra system. Hydra currently supports public data publication, meaning only publishers (and not consumers) need to undergo this bootstrapping process.

C. Decentralized control plane

The decentralized control plane serves as the backbone of Hydra's peer-to-peer federation. At its core, the control plane comprises two essential elements: a synchronized state, known as the "global view," and a distributed decision-making framework. These components collaboratively enable robust, decentralized governance within the Hydra federation.

Synchronized State or Global View: In Hydra, the "Global View" serves as a local database for each node, capturing a comprehensive snapshot of the system's state. Contrary to its name, the Global View is not stored in a universally accessible location; each node maintains its own version. Throughout the system's operation, nodes synchronize their local Global Views by continuously exchanging group messages.

Integral to the Global View is the concept of the "State Vector" [6]. The State Vector signifies a sequence of messages with monotonically increasing sequence numbers assimilated into the Global View, thereby serving as an index for understanding its current state and reconciling any state differences. The Global View encapsulates a variety of information, including details of all participating nodes and specifics of each file. For each file, the Global View identifies the nodes currently possessing the file and those eligible for backup responsibilities. It also includes metadata like file size, origin node, number of copies, and other attributes.

Distributed decision making using "Favor": In Hydra's federated architecture, which spans multiple organizations, the diversity and dynamism of node conditions present unique challenges. Notably, these nodes vary in hardware, storage, bandwidth, and security protocols, all subject to rapid changes. Complicating matters further is the existence of differing administrative domains, making enforcing uniform policies across the federation difficult. Consequently, efficient data replication becomes a multi-optimization problem involving several conflicting constraints such as storage availability, bandwidth, and cost.

To address these complexities, Hydra introduces a mechanism known as "Favor." Each node measures and calculates the Favor number based on its local conditions and replication preferences, encompassing factors like storage availability, network conditions, and data popularity as a composite numeric value. Specifically, nodes with the highest Favor values become the candidates for replicating files that currently fall below their desired degree of replication.

The current Favor calculation uses a weighted formula of three factors – available storage capacity, network bandwidth, and disk read/write speed. However, other more advanced approaches, such as using a multi-objective genetic algorithm to optimize conflicting constraints like storage capacity, network costs, and replication time is also possible [8]. Post-replication, nodes can update their Favor scores to reflect new conditions, such as changes in available storage capacity.

By incorporating Favor, Hydra successfully navigates the challenges arising from node diversity and dynamic conditions. It allows for tailored replication strategies, efficiently allocates resources, and accommodates individual nodes' unique preferences and policies within the federation.

D. Named Content and Service Endpoints

Named Content: In Hydra, the system adopts a publisher-centric approach to naming datasets, offering high flexibility and community-specific customization. For instance, in the field of genomics, it is entirely feasible for the naming to align with the well-established taxonomical structures such as the tree of life [10]. Such a name might look like "`/human/genome/dna/hg38`". Hydra uses these names for all data-related operations such as publication, replication, and access.

Named Services: In Hydra, the architecture employs a streamlined set of named service endpoints, including content publication, deletion, and retrieval. Hydra commonly uses a generic prefix for commands generated by publishers, such as Insert and Delete, and internal communications within the Hydra federation. For example, Hydra could select the prefix "`/Hydra`" or "`/genomics`". In the first case, data insertion and deletion namespaces can be "`/Hydra/insert`" or "`/Hydra/delete`". For internal communications, Hydra uses specific namespaces such as "`/Hydra/group-messages`" and "`/Hydra/heartbeat`" to distribute group messages and heartbeats to all nodes participating in the federation.

IV. HYDRA OPERATIONS

This section discusses the operational aspects of the Hydra federation, elaborating on how Hydra builds services using the building blocks as shown in Fig. 2. Specifically, we discuss the construction and maintenance of a federation, node failure detection and recovery, and procedures for data insertion, automated replication, and data retrieval.

A. Building and maintaining a federation

The Hydra federation structure relies on a multi-step establishment and ongoing maintenance process. Initially, each node undergoes a node bootstrapping phase where it installs the requisite Hydra software and acquires security credentials (i.e., a digital certificate) from the NOC. This ensures secure and authenticated interactions within the federated environment.

Post-bootstrapping, nodes join the Hydra pub-sub namespace that is built using SVS [6]. This connection enables nodes to exchange messages between themselves, including heartbeat and update messages. Heartbeat messages are multicast periodically over the pub-sub namespace (e.g., "`/Hydra`"). They also exchange all update messages over this namespace.

Each node creates and maintains a local database (a.k.a., the Global View) representing its perspective of the federation. This database includes information about other nodes, file attributes, and other metadata. The local databases are synchronized across nodes to achieve a unified state across the federation. When this state changes, nodes send an update, and the other nodes authenticate and apply it to their global views. Hydra assumes an eventual consistency among the global views of the participating nodes.

B. Failure detection and recovery

Heartbeat messages serve as the built-in failure detection mechanism. A failure mode triggers when a node misses three consecutive heartbeats, initiating the recovery protocol. During recovery, each node identifies files needing replication, especially those from the failed node, guided by the global "Favor" metric. If a node ranks highest, it begins replication.

Upon recovery, the node resumes heartbeats while other nodes do not take immediate corrective action. Instead, the reactivated node listens for incoming group messages and updates the ones it missed. If any state data survived, the node calculates the state difference, requesting missing data from other nodes. If no prior state data exists, the node joins as new, updating its state accordingly. This design ensures functional data retrieval during individual node failures as long as one operational node remains in the federation.

C. Data Insertion and Deletion

Within the Hydra framework, both data insertion and deletion follow a secure procedure. When a publisher wants to insert data, the process is initiated by the publisher making contact with a Hydra node. The user sends an Interest and NDN routing brings this Interest to a Hydra node. Concurrently, the user prepares the data for download and listens on a designated data publication namespace. Upon authenticating the user as a legitimate publisher, the node downloads the user's prepared data, completing the insertion and notifies the user and other federation nodes about the new file.

For data deletion, the original publisher contacts a Hydra node. The node authenticates and processes the deletion command. If the file exists locally, the node deletes it and updates its local state, subsequently disseminating a group message to inform the federation. If the file does not exist on the local storage, the node will still update its local state and issue a federation-wide group message, indicating that the file is to be deleted. The Hydra framework operates under the assumption of eventual consistency, ensuring that even if a group message is lost, the system will eventually detect the discrepancy and execute the file deletion.

D. Automated replication

In Hydra's distributed storage framework, data replication is essential for high availability, durability, and efficient data distribution across a geographically dispersed federation of nodes. Replication occurs automatically when a new file is ingested or an existing node fails. On ingestion of a file, a node broadcasts a group message. Other nodes check the replication status of this file, and if below a threshold (the default is three replicas), nodes with the highest Favor lead replication.

Hydra's unique feature is its decentralized approach. Unlike systems like Cassandra with centralized coordination, Hydra shares Favor values among nodes, enabling each to self-identify if they need to participate in replication tasks. Nodes express intent to replicate via group messages. In summary, Hydra's replication is adaptable and resilient. It optimizes data distribution across the federation using decentralized decision-making and dynamic Favor metrics.

E. Data retrieval

In the Hydra framework, any node is capable of handling a user's file retrieval requests, irrespective of whether the node physically stores the file in question or not. To retrieve a file, a user dispatches an Interest bearing the file's name, adhering to the naming schema "`/human/genome/dna/hg38`". Subsequent to this action, three possible scenarios may occur: (a) Should the file not exist within the Hydra ecosystem, the system returns a Negative Acknowledgement (NACK) to the user; (b) If the file is indeed present on the node that was initially contacted, the data corresponding to the Interest is directly returned to the user; (c) In the event that the file exists within the Hydra system but is not on the node first contacted, that node responds with a "Forwarding Hint" that directs the user to another node where the file is stored. The user issues a new series of Interests with this Forwarding Hint that are then channeled via the NDN forwarding mechanism, independent of Hydra, to the node actually containing the data. This initiates a standard Interest/Data exchange procedure for file retrieval.

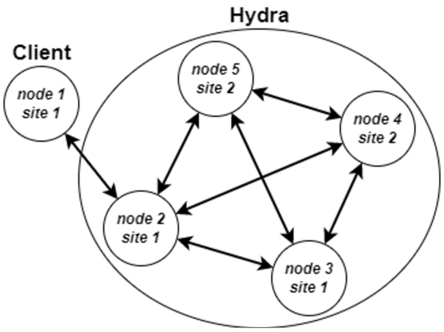

Fig. 3: Hydra topology on FABRIC[5]

V. TRIAL DEPLOYMENT AND EVALUATION

This section describes our preliminary deployment of Hydra on the FABRIC testbed [1]. The primary objective of this deployment is to assess the control plane of Hydra, thereby shedding light on the ramifications of our design choices. As depicted in Fig. 3, our initial setup comprises provisioned nodes on FABRIC, with one node designated as a client and the remainder as Hydra nodes. The Network Operations Center (NOC) is external to FABRIC and not displayed in the figure.

We elected to employ five nodes to showcase Hydra's capabilities preliminarily; four nodes serve as the minimal requisite for replication (the default degree of replication is 3), with an additional node functioning as the client. Each Hydra node had 2 CPU cores, 8 GB RAM, and 20 GB SSD storage. A Layer 2 network was established among these nodes, as represented in Fig. 3.

A. *Evaluation*

It is important to note that as an initial proof-of-concept, our goal was to demonstrate Hydra's capabilities rather than directly compare to other systems.

State Overhead: This experiment evaluates the storage requirements for the local state in Hydra nodes. As Fig. 4 shows, in the initial configuration with 5 nodes and no files, the total state consumed was 32KB, with only 30 bytes attributed to node names. Upon uploading 1,000 files, the total state per node increased to approximately 250KB, of which about 70KB was dedicated to both node and file names. The results suggest that local state requirements in Hydra are relatively low but are subject to increase with longer name lengths. For example, using scientific data names averaging 120-130 characters would add an estimated 130KB to the total state for 1,000 files, resulting in an overall state size of approximately 500KB, which is still small.

Communication Overhead: The lack of a global controller or shared state comes with additional communication overhead. We measure the number of Interests and Data packets over time to quantify this overhead. The messages in this measurement include sync messages, heartbeat messages, and prefix registration and management. Fig. 5 illustrates this communication overhead, revealing that background Interest/-Data exchanges typically generate fewer than 100 Interests and tens of Data packets. However, this overhead is contingent on the number of events within the federation. With 5 Hydra nodes, the packet count escalates to nearly 4,000 Interests and 1,300 Data packets within an hour. To contextualize this, NDN's default packet size is 8800 Bytes, equating to an additional 35MB of network traffic over one hour.

In this experiment, we make several interesting observations. Firstly, there is a disparity between the number of Interests and Data packets. Sync Interests inform nodes of state changes, operating autonomously and remaining unacknowledged, thus not generating Data packets. Fig. 5 also highlights the relatively small scale of the overall state exchange.

Publication Overhead: Since publication in Hydra triggers update messages to the federation, we looked at the overhead of publication as Fig. 6 shows. We start counting the packets when a file is ingested, and an announcement goes out to the federation. The counting stops when replication decisions are made, and the messages confirming the replication decision go out. For 25 file insertions over a 10-minute timespan, we noticed an additional 1,400 Interests. The per-file insertion overhead is approximately 25 Interests, including the insertion command to Hydra, Sync Interests, and background traffic. Note that we are measuring the control overhead here, not the actual data overhead. The data overhead is equal to the number of replications.

Replication decision: In Hydra, the default degree of replication is 3. Hydra uses either new file insertion or node failure to trigger replication. This experiment quantifies how long it takes to make these replicate decisions. In this experiment, we measured the time between detecting a new file (or a node failure) and the time for the other two nodes to make the replication decision. In our experiment, after the group message went out to the nodes announcing a new file, it took the first node 8.344±14.40 milliseconds to start replicating the content. The second node needed 38.404±13.24 ms to start replication. The exact time to start replication depends on factors such as distance to the original node, congestion, and node load, but we observe that replication starts very quickly under normal operating conditions.

Node joining and failure detection: In Hydra, a heartbeat message goes out to the federation once a new node joins. On the other hand, node failures are detected through the lack of heartbeats. In this test, we added nodes to the federation to quantify the time it takes to get the node to join it. We also randomly failed nodes to quantify how long the other nodes took to detect the failure.

We found after ten runs that the time for a new node to join the federation is 55.428±0.238 seconds. Note that this time varies based on when the heartbeat goes out to the group. Each node sends out a heartbeat every 15 seconds. Each node also waits for three heartbeats before updating the local state. The total time includes three heartbeats, time to register prefixes, and start up the Hydra software framework. Failure detection takes longer by design. In our experiments, after ten runs, we found the average failure detection time was 94.608± 5.38 seconds. The failure detection routine in Hydra runs every 30 seconds to accommodate any possible delay affecting the heartbeat. The total time here includes three heartbeat detection cycles and time to process the failure.

Distributed decision making: We mentioned previously how Hydra nodes make replication decisions based on the

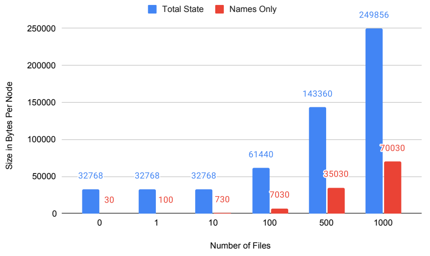

Fig. 4: State size vs. Number of Files Published

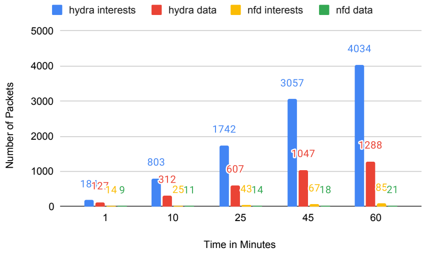

Fig. 5: Network Overhead of Hydrawithout any Client Interaction

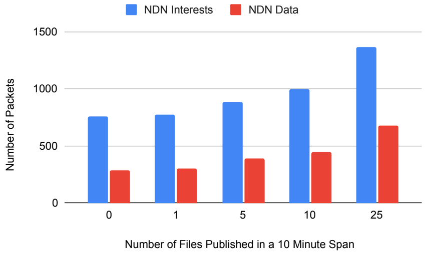

Fig. 6: Network Overhead of Hydrawith Client Interaction

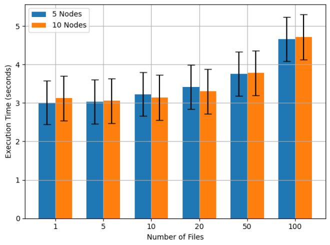

Fig. 7: Favor calculation Time

Favor parameter. This experiment aimed to evaluate the execution times of the Favor calculation process using different numbers of nodes (5 and 10) under a bandwidth constraint of 100 Gbps. The process involved calculating the Favor using a greedy algorithm for establishing the replication order and calculating multi-objective optimization. Fig. 7 shows the time required for Favor calculation for replicating 1, 5, 10, 20, 50, and 100 files. For both 5 and 10 nodes, the execution times remain relatively stable as the number of files increases. Note that this value is calculated and stored separately in Global View and does not affect Hydra's operation.

VI.DISCUSSIONS AND LESSONS LEARNED

Advancing FAIR Principles through Secure Decentralization: In light of the technical specifications described in the earlier sections, we examine the implications of Hydra's architecture for scientific workflows, particularly in terms of data publication, access, and adherence to FAIR [9] principles (Findability, Accessibility, Interoperability, and Reusability).

Data Publication and Findability: Traditional centralized solutions for data storage and retrieval often constrain data publication to specific locations. In such scenarios, the findability of data largely depends on metadata that is centrally managed. Hydra's decentralized architecture, in contrast, liberates data from being tied to specific physical locations. This is achieved through name-based data retrieval. Named data becomes an integral identifier, making the data findable irrespective of its storage location, thereby advancing the first aspect of FAIR principles, i.e., Findability.

Data Access and Accessibility: Hydra's use of name-based access mechanisms extends considerable benefits in data access. The stateful network transport and in-network caching allow for efficient data retrieval without putting undue load on individual nodes. This has significant implications for scientific workflows, where large data sets are often accessed by multiple users. Such a system design directly addresses the second aspect of FAIR principles, which is Accessibility.

Interoperability and Reusability: Hydra's decentralized architecture promotes data interoperability and reusability. By utilizing data names for all operations, Hydra ensures that data formats and structures are transparent and interoperable. Moreover, the direct signing of data by the producer in Hydra ensures data integrity and enables content from anywhere, making it easier for researchers to reuse the data. These mechanisms directly align with the Interoperability and Reusability aspects of the FAIR principles.

Security and Provenance in Decentralized Scientific Workflows: Hydra's decentralized trust model boosts security by letting users set their trust anchors. It also supports publisher authentication for secure data management, reducing central authority bottlenecks and ensuring data integrity. This aligns with FAIR data principles in a secure framework.

Scalability and Resiliency in Scientific Workflows: Hydra is designed to be both scalable and resilient, two features crucial for scientific workflows. The decentralized architecture allows the system to scale horizontally, accommodating an increasing number of nodes and data requests. On the resiliency front, the decentralized data sharing and lack of central control eliminates single points of failure, thus ensuring uninterrupted data availability even when certain nodes are unavailable.

Limitations and Implications of Eventual Consistency in Hydra: In this section, we lay out some limitations of the Hydra system. First, Hydra assumes a federation of organizations has some knowledge of which nodes and users can join the Hydra federation. This knowledge (DNS names for nodes and email addresses of users) is built into the NOC.As mentioned earlier, the NOC's role is limited to certificate distribution, and it does not exercise any other control over the Hydra infrastructure. As such, the NOC is not a single point of failure in a Hydra federation.

Second, we also assume Hydra publishes publicly available open data. We are working on building an access control model for the consumers but this is not yet integrated with the system.

Third, Hydra operates under an eventual consistency model, implying that data replication and state synchronization occur on a best-effort basis rather than in real time. The implications of this model are twofold. Firstly, regarding data insertion and replication, any delays in the ingestion notification can result in file replication remaining below the preferred degree. Nevertheless, the file becomes accessible to consumers once the global view is synchronized. Fig. 5 illustrates that this state exchange is typically minimal, suggesting that synchronization should be quick under normal operational conditions. In the event of congestion or network partition, an unsynchronized state can lead to the ingested file remaining inaccessible and unreplicated until state synchronization.

Lastly, we have yet to comprehensively grasp the ramifications of autonomous replication decisions on the entire system. Can a scenario arise in which a file remains unreplicated not because of resource constraints but rather due to policy limitations? If such a situation does exist, how should we address it? We are currently exploring answers to these questions.

VII. RELATED WORK

There have been several attempts to create distributed data repositories in the past, and some of these have been successfully deployed. Popular distributed databases such as Cassandra, Bigtable, and Dynamo [7] and other similar solutions can store large amounts of data and perform replication functions. These systems are tightly coupled, generally require significant manual configuration and maintenance, and, most importantly, require a single administrative control for the configuration of replication and other system functions – a model that is unfit to serve a community of scientists, where individual machines may be owned by different parties and require some degree of autonomy in their operations.

Other distributed data management infrastructures also exist in scientific communities, including Xrootd, iRods, Rucio, and Globus [11]. These solutions hide the complexity of a location-independent infrastructure over TCP/IP at the application layer by creating a location-transparent overlay. However, they still need to maintain data locations which makes them complex and requires substantial manual configuration.

There have been a few incarnations of storage repositories over NDN such as repo-ng, Fast Repo for NDN-RTC streams, NDNts for web applications, and ndn-python-repo. These existing NDN based repositories are single-instance implementations of storage that can be accessed over the network, but not a distributed storage system.

VIII. CONCLUSION AND FUTURE WORK

In addressing the challenges posed by big data scientific research on networked systems, Hydra offers a secure, scalable, and resilient storage service by leveraging a decentralized federation of individual user-provided storage servers. Grounded on Name Data Networking (NDN), Hydra exemplifies that loosely coupled, name-based systems can be both lightweight and robust. This work has afforded us valuable insights into the intricacies of crafting a secure, data-centric distributed network without micromanaging every individual node.

As we continue to refine Hydra, several areas of improvement are under exploration. Key among these is the optimization of the data plane for higher throughput via NDN-DPDK integration. Additionally, we aim to enhance data retrieval performance through congestion control and module tuning. Another focus is the dynamic adjustment of the favor parameter based on near real-time performance metrics. Benchmarking against existing solutions will be another important next step.

In contrast to the growing centralization in IP networks, Hydra and NDN pave the way for a decentralized approach. Leveraging NDN's data-centric networking model, which includes features like in-network caching and multicast delivery, our work opens up new possibilities for empowering end users. This research serves as a cornerstone for creating a more democratic and decentralized scientific federation and has the potential to revolutionize secure, decentralized storage, making it not just viable but highly effective.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation under Grant No OAC-2126148.

REFERENCES

[1] BALDIN, I., NIKOLICH, A., GRIFFIOEN, J., MONGA, I. I. S., WANG, K.-C., LEHMAN, T., AND RUTH, P. Fabric: A national-scale programmable experimental network infrastructure. *IEEE Internet Computing 23*, 6 (2019), 38–47.

[2] EDWARDS, P. N., MAYERNIK, M. S., BATCHELLER, A. L., BOWKER, G. C., AND BORGMAN, C. L. Science friction: Data, metadata, and collaboration. *Social Studies of Science 41*, 5 (2011), 667–690. PMID: 22164720.

[3] LIU, S., PATIL, V., YU, T., AFANASYEV, A., FELTUS, F. A., SHANNIGRAHI, S., AND ZHANG, L. Designing hydra with centralized versus decentralized control: A comparative study. In *Proceedings of the Interdisciplinary Workshop on (de) Centralization in the Internet* (2021), pp. 4–10.

[4] OGLE, C., REDDICK, D., MCKNIGHT, C., BIGGS, T., PAULY, R., FICKLIN, S. P., FELTUS, F. A., AND SHANNIGRAHI, S. Named data networking for genomics data management and integrated workflows. *Frontiers in Big Data 4*, 1 (2021), 16.

[5] PRESLEY, J., WANG, X., BRANDEL, T., AI, X., PODDER, P., YU, T., PATIL, V., ZHANG, L., AFANASYEV, A., FELTUS, F. A., ET AL. Hydra–a federated data repository over ndn. *arXiv preprint arXiv:2211.00919* (2022).

[6] SHANG, W., AFANASYEV, A., AND ZHANG, L. Vectorsync: Distributed dataset synchronization over named data networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking* (New York, NY, USA, 2017), ICN '17, Association for Computing Machinery, p. 192–193.

[7] STANSBERRY, D., SOMNATH, S., BREET, J., SHUTT, G., AND SHANKAR, M. Datafed: Towards reproducible research via federated data management. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)* (Los Alamitos, CA, USA, dec 2019), IEEE Computer Society, pp. 1312–1317.

[8] WANG, X., AI, X., FELTUS, F. A., AND SHANNIGRAHI, S. Gnsga: A decentralized data replication algorithm for big science data. In *2023 IFIP Networking Conference (IFIP Networking)* (2023), pp. 1–9.

[9] WILKINSON, M. D. E. A. The fair guiding principles for scientific data management and stewardship. *Scientific Data 3*, 1 (Mar 2016), 160018.

[10] WOLF, Y. I., ROGOZIN, I. B., GRISHIN, N. V., AND KOONIN, E. V. Genome trees and the tree of life. *TRENDS in Genetics 18*, 9 (2002), 472–479.

[11] WU, Y., MUTLU, F. V., LIU, Y., YEH, E. M., LIU, R., IORDACHE, C., BALCAS, J., NEWMAN, H., SIRVINSKAS, R., LO, M., SONG, S., CONG, J., ZHANG, L., TIMILSINA, S., SHANNIGRAHI, S., FAN, C., PESAVENTO, D., SHI, J., AND BENMOHAMED, L. N-dise: Ndn-based data distribution for large-scale data-intensive science. *Proceedings of the 9th ACM Conference on Information-Centric Networking* (2022).

[12] ZHANG, L., AFANASYEV, A., BURKE, J., JACOBSON, V., CLAFFY, K., CROWLEY, P., PAPADOPOULOS, C., WANG, L., AND ZHANG, B. Named data networking. *ACM SIGCOMM Computer Communication Review 44*, 3 (2014), 66–73.

[13] ZHANG, Z., YU, Y., AFANASYEV, A., AND ZHANG, L. NDN certificate management protocol (NDNCERT). Technical Report NDN-0050, NDN, Apr. 2017.