



Elettra Sincrotrone Trieste

IPFS: the perspective storage infrastructure for scientific data

Andrey Vukolov, ICT Group, ELETTRA

24/09/2020

PaN data: storage backends landscape

Centralised / server-based

- Globus Connect
- IBM GPFS
- IBM Elastic Storage
- Hitachi Vantara
- OODRIVE ActiveCircle

Decentralised / grid-based

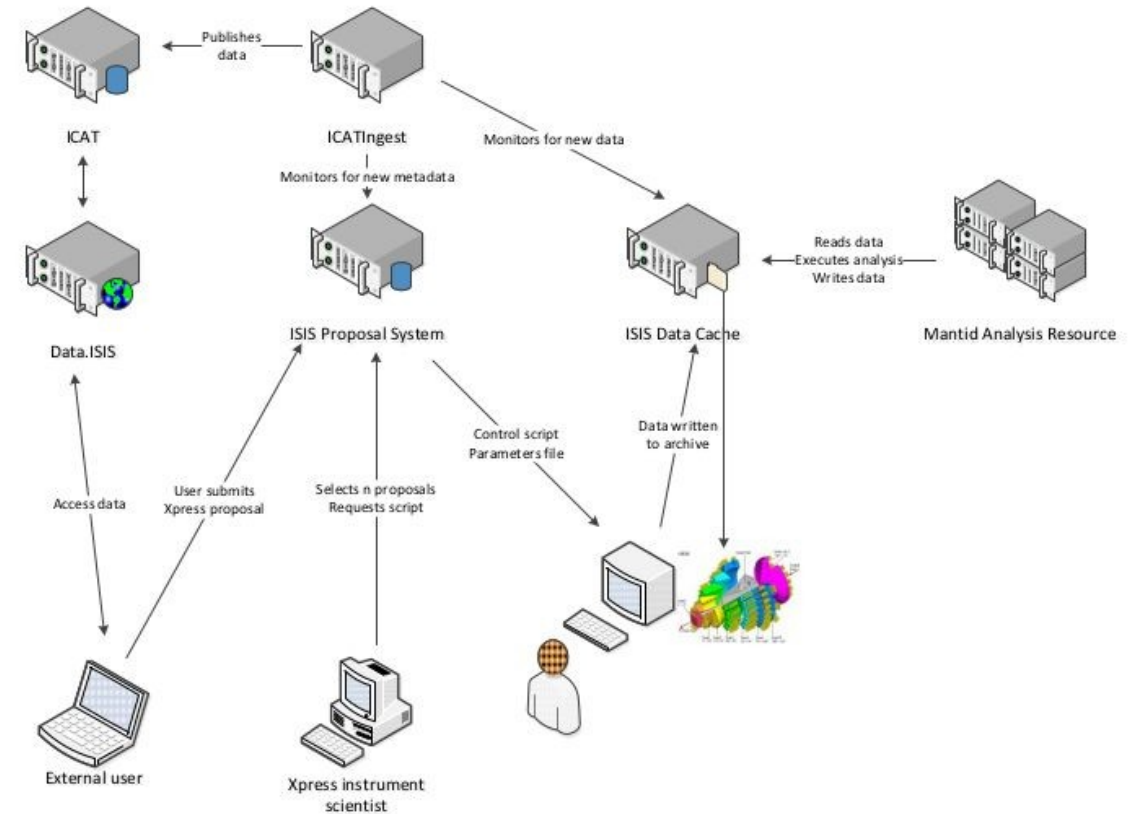
- Lustre
- iRODS
- CEPH
- GlusterFS
- Hash-based (GNUtella, BitTorrent)

Challenges of PaN data storage backend landscape

- **Monolithic / centralised architecture;**
- Non-compliance of the centralised data flow model with FAIR;
- Data integrity provenance for large amounts;
- Metadata incompatibility;
- Data flow distribution inside the networks;
- Addressing gaps;
- Proprietary software inside the workflow.

Problem: centralised architecture

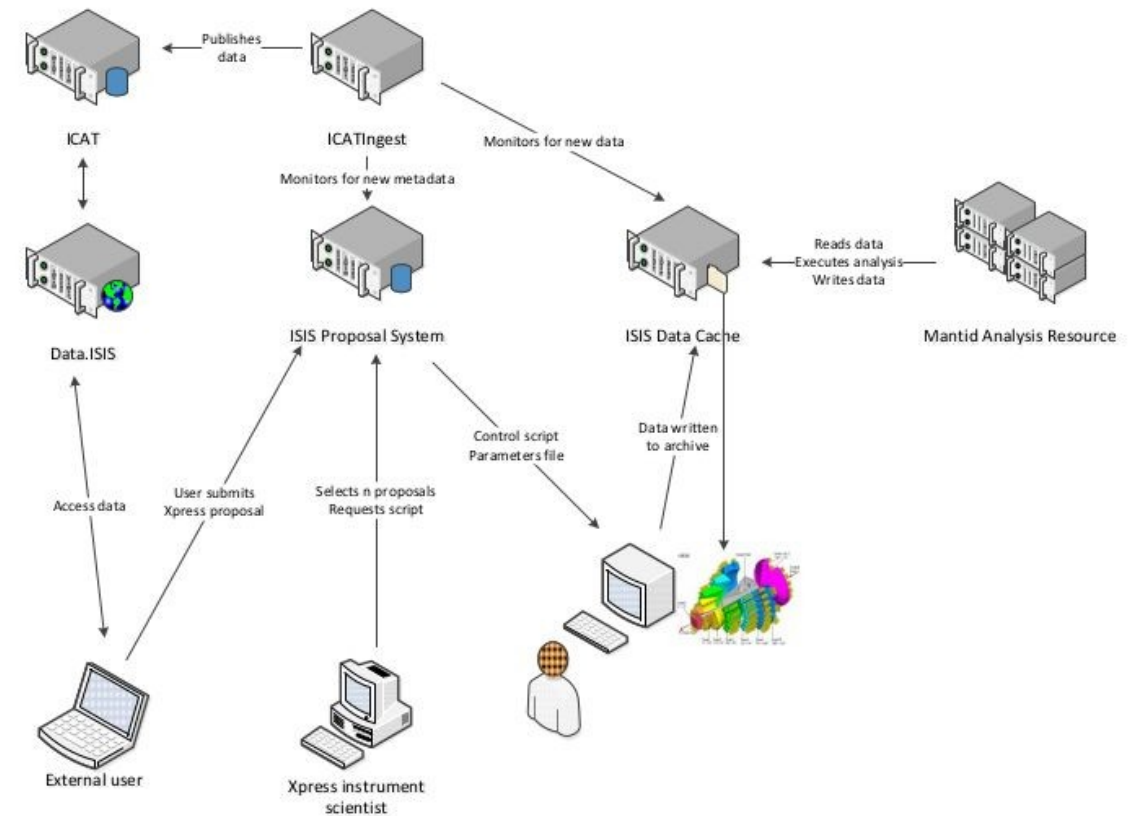
- Every single node in the schema is single data processing point;
- The data are in constant flow between instrument, storage facilities and user workstations;



ISIS data infrastructure (from the [PaNData Continuum Model](#))

Problem: centralised architecture

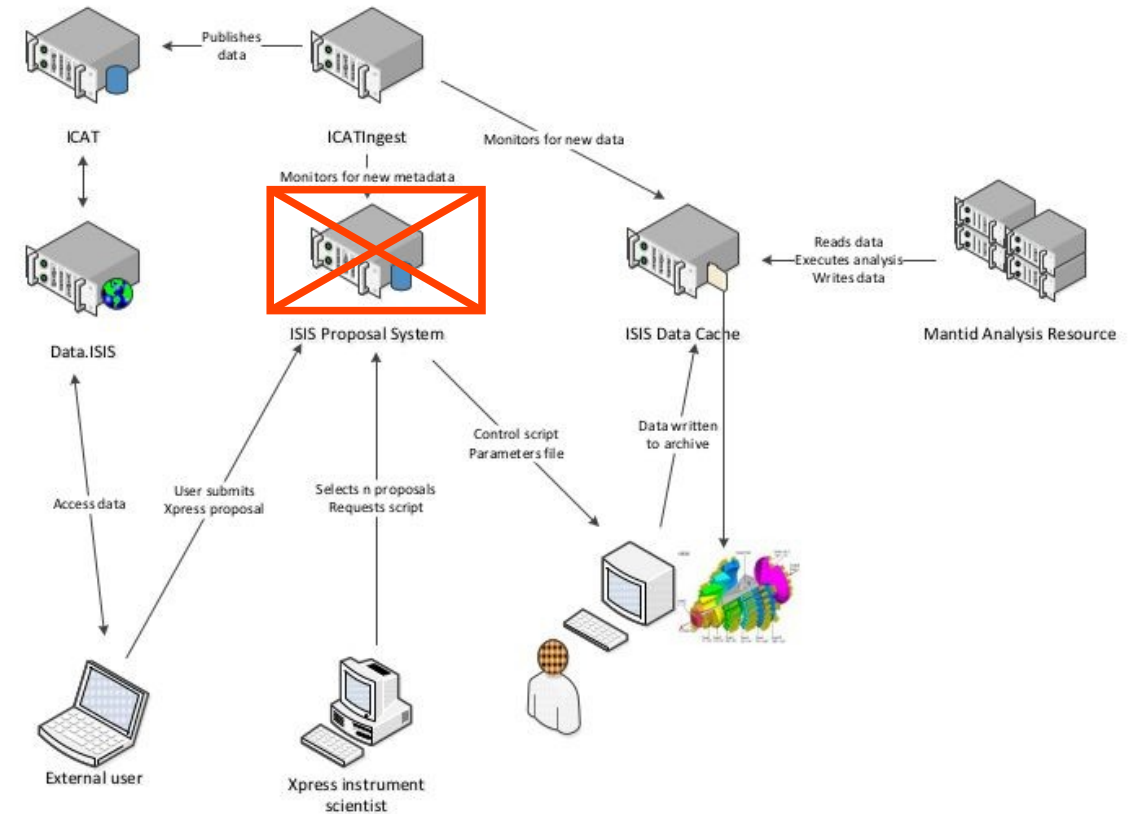
- Every single node in the schema is single data processing point;
- The data are in constant flow between instrument, storage facilities and user workstations;
- Each node is a separate **point of failure** because any technical failure leads to data loss;
- The transmission tasks are concurrent in any case of transferring large volumes of data



ISIS data infrastructure (from the [PaNData Continuum Model](#))

Problem: centralised architecture

- Every single node in the schema is single data processing point;
- The data are in constant flow between instrument, storage facilities and user workstations;
- Each node is a separate **point of failure** because any technical failure leads to data loss;
- The transmission tasks are concurrent in any case of transferring large volumes of data



ISIS data infrastructure (from the PaNData Continuum Model)

Challenges of PaN data storage backend landscape

- Monolithic / centralised architecture;
- **Non-compliance of the centralised data flow model with FAIR;**
- Data integrity provenance for large amounts;
- Metadata incompatibility;
- Data flow distribution inside the networks;
- Addressing gaps;
- Proprietary software inside the workflow.

Challenges of PaN data storage backend landscape

- Monolithic / centralised architecture;
- Non-compliance of the centralised data flow model with FAIR;
- **Data integrity provenance for large amounts;**
- Metadata incompatibility;
- Data flow distribution inside the networks;
- Addressing gaps;
- Proprietary software inside the workflow.

Challenges of PaN data storage backend landscape

- Monolithic / centralised architecture;
- Non-compliance of the centralised data flow model with FAIR;
- Data integrity provenance for large amounts;
- **Metadata incompatibility;**
- Data flow distribution inside the networks;
- Addressing gaps;
- Proprietary software inside the workflow.

Challenges of PaN data storage backend landscape

- Monolithic / centralised architecture;
- Non-compliance of the centralised data flow model with FAIR;
- Data integrity provenance for large amounts;
- Metadata incompatibility;
- **Data flow distribution inside the networks;**
- Addressing gaps;
- Proprietary software inside the workflow.

Challenges of PaN data storage backend landscape

- Monolithic / centralised architecture;
- Non-compliance of the centralised data flow model with FAIR;
- Data integrity provenance for large amounts;
- Metadata incompatibility;
- Data flow distribution inside the networks;
- **Addressing gaps;**
- Proprietary software inside the workflow.

Problem: addressing

- The open data repositories usually support uploading and downloading data through **URI** which is considered as metadata;
- The instruments and storage systems expose the data through interfaces incompatible with standard URI;
- Authorization, versioning and access control are the special tasks that require internal **content-based** addressing for the data inside the repository;
- The existing **path-based** addressing model is subject to preserve because of high presence inside the community (NeXuS, HDF).

Challenges of PaN data storage backend landscape

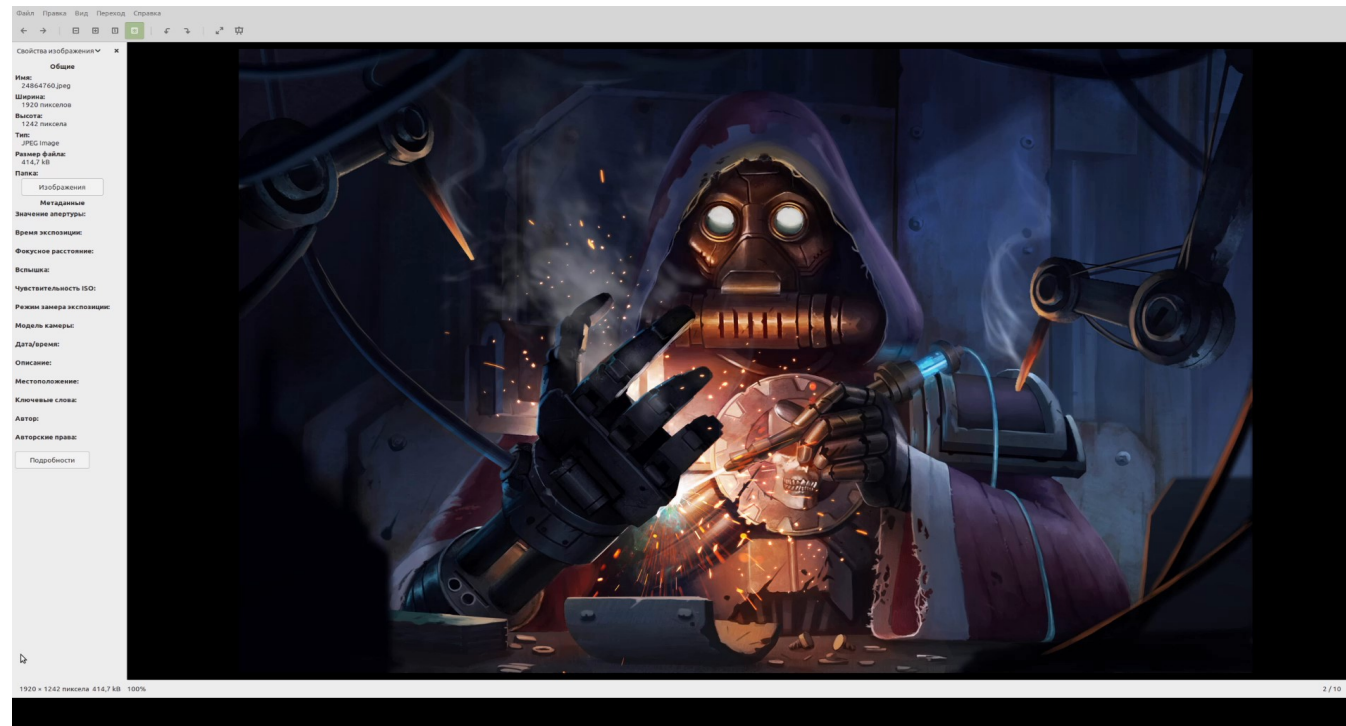
- Monolithic / centralised architecture;
- Non-compliance of the centralised data flow model with FAIR;
- Data integrity provenance for large amounts;
- Metadata incompatibility;
- Data flow distribution inside the networks;
- Addressing gaps;
- **Proprietary software inside the workflow.**

Key features landscape for the storage backend suitable for FAIR data sharing in PaN community

- Infrastructure-independent **addressing model** also implementing content-based addressing;
- Residual **data persistence** with automated recovery possibilities;
- Autonomous **scalability** with minimized reconfiguration of nodes (“Hot swap”);
- Weak sensitivity to networking issues between the nodes;
- Optimized bandwidth utilization mechanisms;
- Absolute **persistence** for an exposed ID of the given storage entity;
- Cross-platform **client interface** and/or API compatible with standardized operating environment;
- **Default** configuration (or possibility to create a shared one) enough to start the node in the given deployment (“Fire&Forget”).

IPFS: distributed storage system

InterPlanetary File System (IPFS) splits the data into the blocks joined into direct acyclic graphs (DAG) which can be distributed across the network



IPFS: licensing

- Source code of the daemon is published under regulations of double **Apache 2.0 / MIT license**.
- These licenses are both permissive, they allow to freely distribute and modify the source code without any limitation to jurisdiction in which the code or the corresponding intellectual property is located.
- The only condition to apply the licensing model to the source code derived from IPFS is to change the name.
- Any possible conflict of jurisdictions could be easily avoided from the legal point of view, by forking and building IPFS source code with attribution which is compatible with **Apache License 2.0** and **Creative Commons Attribution 3.0 Unported license**.

IPFS addressing model: CID

CID: globally-unique, immutable, ASCII encoded multihash identifying any part of IPFS data flow.

- All data are formatted using self-representing specifications ([multibase](#), [multicodec](#), [multihash](#));
- Character prefixes are codetable-agnostic;
- Every DAG node has its own CID and supposed to be a separate (nested) graph;
- DAG representation is self-balancing distributed hash table structure ([kademlia](#)/BitTorrent DHT).

Example:

base58btc encoded version (CID v0):

QmcrWspiMXVAfD2UsdwxgHdf9hphPiN4J3ebF7j6ns4vK3

expands to base32 CID v1:

|b|afybeigxvixv5umcyhmxnfeb5bap3pqvdeeuerkf4cq2livoqhi j6jm7qi

Encoding: base32 = 'b' literal prefix

Version: CID v1 = 0x01

Content type: Protobuf-based DAG node = 0x70

Hash function: SHA2-256 = 0x12

Hash size: 32 bytes / 256 bits = 0x20

Hexadecimal-encoded hash digest: =

= 0xd7aa2f5ed182c1d9769481e840fdb151909424545e0a1a5a2ae81d09f259f82

IPFS addressing model: network

IPFS utilizes `multiaddr` specification to replace URI for internal purposes.

Example: BMSTU public node gateway address for URI `https://ipfs.robotics.bmstu.ru/ipfs/`:

`/dnsaddr/ipfs.robotics.bmstu.ru/tcp/443/https`

Network search protocol	Network search namespace	Transport layer protocol	Port	Application layer protocol
<code>/dnsaddr</code>	<code>/ipfs.robotics.bmstu.ru</code>	<code>/tcp</code>	443	<code>/https</code>
DNS over both IPv4/IPv6	Fully-qualified DNS name	TCP/IP		HTTP over SSL gateway

The multiaddresses are recursive. They can be used to specify routes over relays.

Example: node address to access through public relay only:

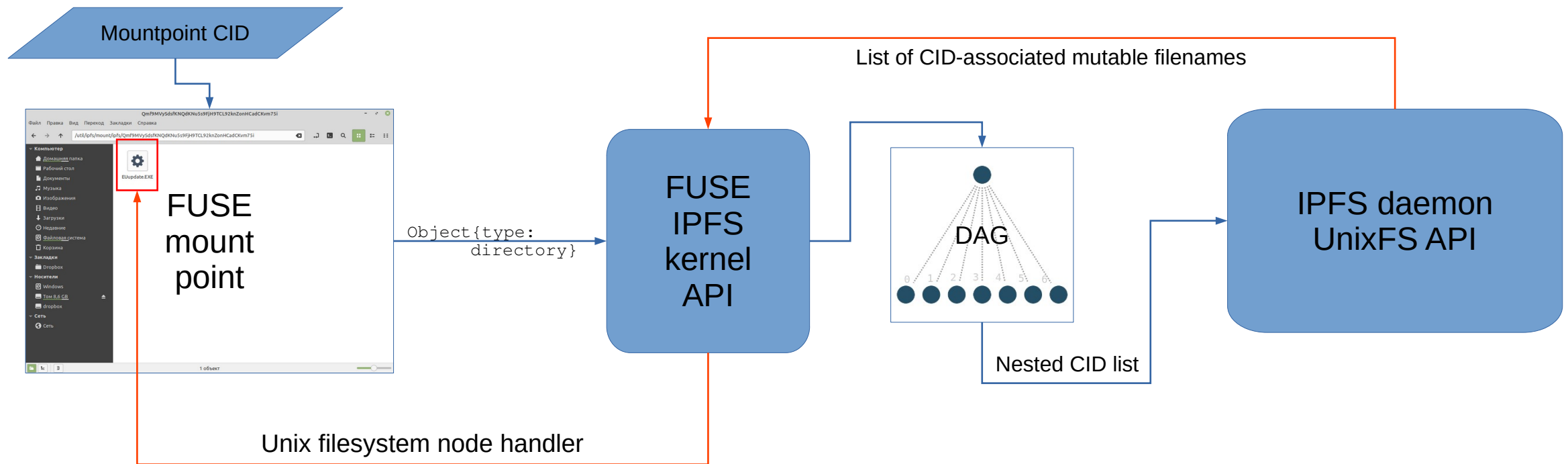
`/ip4/147.75.70.221/tcp/4001/p2p/Qme8g49gm3q4Acp7xWBKg3nAa9fxZ1YmyDJdyGgoG6LsXh/p2p-circuit/p2p/QmSg6L7KwZL1618ANhMxJErxBTNLUheUYbB75Rv36vceYy`

contains following multiaddresses (*relay* address + *peer* address):

- `/ip4/147.75.70.221/tcp/4001/p2p/Qme8g49gm3q4Acp7xWBKg3nAa9fxZ1YmyDJdyGgoG6LsXh`
- `/p2p-circuit/p2p/QmSg6L7KwZL1618ANhMxJErxBTNLUheUYbB75Rv36vceYy`

IPFS addressing model: Unix/FUSE

IPFS running as a daemon, can attach file of directory Unix attributes to the data associated with given CID. Then the local FUSE mount point can be provided by the daemon to the system. The filenames in FUSE mountpoint are virtual, mutable values, snapped temporarily to corresponding CIDs.



IPFS: pinning and providing

- The data blocks uploaded into IPFS DAG are exposed to DHT by their CIDs. The data are stored in the key-value noSQL database (`flatfs`, `badgerDB`, `LevelDB`);
- DHT marks CIDs as **provided** ones when the corresponding data are explicitly stored on the local machine or the relay from which the data can be downloaded is accessible;
- DHT can mark the given CID as **wanted**. The corresponding data will be downloaded whenever they become available at any node of the whole network, time for waiting is unlimited;
- The given CID could be marked as **pinned** one by the user. Pinning marks the CID and all corresponding DAG nodes as the data that should be stored locally on the machine where the pinning operation was initiated;
- The given element of data (for example, file) stays **persistent** and **immutable** until the corresponding root CID is pinned by at least one machine and there are existing download requests through significant (indeterminable at the moment) period of time.

IPFS: data presence and persistence

DAG CID state	Pinned	Wanted	Available	Discarded
<i>Local data state</i>				
<i>Stored</i>	Network-wide persistent	Network-wide available	Network-wide persistent	Network-wide available
<i>Requested</i>	Network-wide persistent	Network-wide available	Network-wide available	Available at first accessible relay
<i>Idle</i>	Network-wide persistent	Network-wide available	Network-wide available	Network-wide available
<i>Discarded</i>	Network-wide persistent	Available at first accessible relay	Available at first accessible relay	Unavailable with given CID

IPFS: deployment variants

- Standalone/desktop:
The daemon announcing only `libp2p` multiaddresses to resolve `ipfs://` protocol prefix on the browser level. The `flatfs` storage backend is used by default. Daemon is run by browser extension.
- Server/container:
The machine-specific single daemon announcing publicly exposed IP address for `libp2p` swarm, partially involved as relay. The `badgerDB` storage backend is recommended. The daemon is run under governance of kernel initialization system.
- Cluster: server daemons organized into the redundant structure.
 - Orchestration solution maintaining the list of CIDs pinned on the given set of the machines with redundant storage;
 - Built-in *web of trust* mechanism to maintain voluntarily supported cluster sharing the space to store encrypted data **pinned** on each involved machine.

IPFS: initial testing protocol

Test cases (24 rounds per case, all data CIDs unpinned on receiver's side):

1. Download of a single-file dataset ~3GiB size from HTTP gateway onto the local machine.
2. Download of a single-file dataset ~3GiB size from IPFS command-line interface onto the local machine with compression turned off.
3. Simultaneous download of two single-file datasets with different sizes (1GiB + 3 GiB) with external IPFS daemon, command-line interface and IPFS protocol parser (IPFS Companion + Firefox 78.0.0).
4. Transmission stability test - playing video with high (13 MBit/s) flow rate from IPFS through HTTP in a case when the video itself was not pinned on the same machine with the gateway.

Transmitter / HTTP gateway	Data receiver	Spare seeder / repeater
<ul style="list-style-type: none">• IPFS 0.7.0-dev standalone, systemd, x64• Octacore Xeon E5440 CPU, 6.2GB RAM• 1 GBit/s Ethernet• BadgerDB storage backend, 256GiB space limit	<ul style="list-style-type: none">• IPFS 0.7.0-dev standalone, systemd, x64• Octacore Intel Core i7-6700 CPU, 16GiB RAM• 1 GBit/s Ethernet• BadgerDB storage backend, 8GiB space limit	<ul style="list-style-type: none">• IPFS 0.7.0-dev standalone, systemd, i386• Intel Pentium IV v2 1.8 CPU, 2GiB RAM• 200 MBit/s IEEE 802.11an radio• flatfs storage backend, 128GiB space limit

IPFS: test results

Test case, transfer type	3GiB, single file, binary dataset, HTTP gateway [1]	3GiB, single file, binary dataset, IPFS P2P [2]	1GiB + 3 GiB, double-threaded IPFS P2P, protocol parser [3]	Video transfer, 13 Mbit/s MKV TS, IPFS relay <-> HTTP gateway [4]
Maximum speed [MiB/s]	5.8	10.2	4.1	1.625 (video limit)
Average speed [MiB/s]	3.2	5.1	1.7	1.625 (stable), controllable
Failure count (for 24 rounds)	2	0	0	0
Root CID state [Transmitter]	Pinned	Pinned	Pinned	Wanted
Root CID state [Spare]	Pinned	Pinned	Pinned + Idle	Pinned
Local data state [Transmitter]	Stored	Stored	Stored	Requested
Request type [Receiver]	HTTP [GET]	IPFS [pin add]	IPFS [get]	HTTP [GET]
Local data state [Receiver]	None	Requested	Requested	Cached

Discussion: key points

- In case when the HTTP gateway and URI-based addressing used, the end-user have no need to install any kind of additional software;
- IPFS itself does not provide authentication and authorization by itself, it is definitely the data storage backend;
- On the network level, IPFS provides scalability out of the box. If the feature is not explicitly turned off, the daemon discovers peers automatically and permanently seeds the data into the network;
- OS-compatible client interfaces are built-in within the daemon at least for POSIX-compatible environment;
- Cluster deployments allow to build URI-exposing clusters that aren't based on the given physical infrastructure located at the same networking endpoint, including infrastructures based on HTTP transceivers;
- IPFS allows RIs to dedicate the known amount of resources to shared data without restrictions.

Discussion: security

Possibilities to implement access control for IPFS provided data:

- CID-oriented:
 - Wrapping CID with URI corresponding to specially organized UnixFS structures (dedicated virtual directory as root CID);
 - Rehashing of the given CID;
 - Closing set of unpublished CIDs inside private [IPFS Cluster](#) deployment;
 - Publishing of temporarily available CIDs with limited lifetime from `ipfs name --ttl` subroutines;
- External:
 - Gateway-side URI-based access control with authentication database;
 - Customized HTTP redirection.

Discussion: CID persistence

CID is internally persistent identifier for the given data allowing direct hash-based content addressing model. In accordance with set of criteria defined in [EOSC PID Policy](#):

- CID specifications defined with persistent, self-describing syntax;
- CID for the given storage entity can be considered **globally-unique** until mass collision will be proven for the corresponding hash function;
- CID is completely content-driven. IPFS has **no central authority** to issue or to assign CIDs excepting the virtual directories;
- CID itself supports neither namespaces nor versioning. It is, however, possible using URI-based addressing model and/or UnixFS API;
- CID is “resurrectable” while the associated data exists, but unavailable inside IPFS, so absolute reproducibility is maintained out-of-the-box.

Discussion: CID persistence

CID is internally persistent identifier for the given data allowing direct hash-based content addressing model. In accordance with set of criteria defined in [EOSC PID Policy](#):

- CID hides object location from the user because of DAG integrated into the DHT resolver and `libp2p`;
- Dependencies between CIDs could be defined and resolved in terms of DAG and IPNS;
- CID can be defined as reversible link to any other type of Persistent Identifier defined in EOSC policy;
- The given CID can be added to the DAG associated with another CID as link producing a separate DAG object and IPFS tree.

Example: CID as Persistent Identifier

Input data / generated CID (DOI and URL are treated as strings):

DOI 10.5281/zenodo.4014811 \Leftrightarrow QmejBCiBWeYqKfkZHr1KCokhAwXz9dbbpFshYb3KphxAes

URL https://zenodo.org/record/4014811 \Leftrightarrow QmVXpUKy2iX9fcrm4fNegCaatpMLoiR9Do9KdF7fnp4ino

Filename: 857641_D2.1-DraftDataPolicyFramework.pdf \Leftrightarrow Qmbwh69pjeFLv4YfR1vWpgD1RVd6EHcLCMbvgBBdsoEKWu

PDF file 857641_D2.1-DraftDataPolicyFramework.pdf \Leftrightarrow QmSX1jSPZBNiBKATuPUMnHAauShPbRapCPwAp2NTsX9nwM

QmejBCiBWeYqKfkZHr1KCokhAwXz9dbbpFshYb3KphxAes as **old root CID**

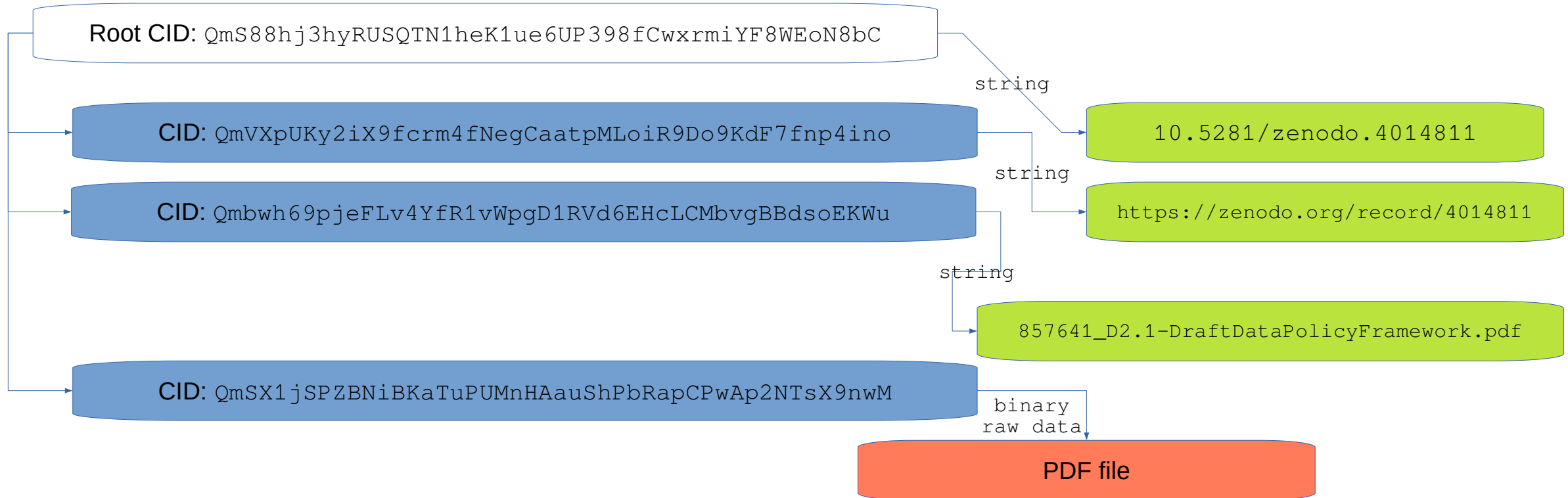
```

|
+ QmVXpUKy2iX9fcrm4fNegCaatpMLoiR9Do9KdF7fnp4ino as 'url'
  |
  | QmXSiQm7gzwMFPMwi8g8c1j1FfBbD1EApyXKHGNCsKW8RL
    |
    + QmSX1jSPZBNiBKATuPUMnHAauShPbRapCPwAp2NTsX9nwM as 'file'
      |
      | QmV5zsjB1ypEapCWDCGRF7SaxXFez1D4bH1raYowxXbEmb
        |
        + Qmbwh69pjeFLv4YfR1vWpgD1RVd6EHcLCMbvgBBdsoEKWu as 'filename'
          |
          | QmS88hj3hyRUSOTN1heK1ue6UP398fCwxrmiYF8WEoN8bC as new root CID

```

Example: CID as Persistent Identifier

New root CID: `QmS88hj3hyRUSQTN1heK1ue6UP398fCwxrmiYF8WEoN8bC` addresses named virtual objects linked into the DAG. In this example the structure contains string-typed objects that can not be represented as files in UnixFS API.



Conclusions

- IPFS provides residual data persistence with recovery possibilities;
- IPFS CID could be considered as the entity similar to Persistent Identifier in terms of EOSC documentation;
- IPFS provides persistent addressing model adaptable to URI schema and “resurrectable” content identification;
- The IPFS daemon provides autonomous scalability with weak sensitivity to networking issues without reconfiguration or with possibility to create shared configuration for cluster deployment;
- OS-compatible client interface exposing the registered data is available for FUSE;
- IPFS allows to control access to the given data by itself publishing CIDs with limited lifetime;
- IPFS could be easily integrated into storage workflow and data cataloguing;
- The redundant storage model allows to share the data with copying-on-demand reducing network loads.

Acknowledgements

- **Christopher Reynolds**, Diamond Light Source, United Kingdom;
- **Abigail McBirnie**, STFC/UKRI, United Kingdom;
- **Zdeněk Matěj**, MAX IV, Sweden;
- **Alun Ashton**, Paul Scherrer Institut, Switzerland;
- **Idrissou Chado**, Synchrotron SOLEIL, France;
- **Adara Carrion Gallegos**, Synchrotron ALBA, Spain;
- **Robert Rosca**, Synchrotron DESY, Germany;
- **Roberto Passuello**, ELETTRA Sincrotrone Trieste, Italy;
- **Stuart Pullinger**, STFC/ICAT, United Kingdom
- **Paul Millar**, Synchrotron DESY, Germany;
- **Marco DeSimone**, ELETTRA Sincrotrone Trieste, Italy;
- **Alejandra Gonzalez-Beltran**, STFC/UKRI, United Kingdom

Thank you for your attention!