Science of Computer Programming 238 (2024) 103174

Contents lists available at ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

Original software publication

IPFS requested content location service [☆]

Pedro Ákos Costa [a,*], João Leitão [a], Yannis Psaras [b]

[a] NOVALINCS & NOVA University of Lisbon, Lisbon, Portugal
[b] Protocol Labs, United States of America

ARTICLE INFO

Keywords:
IPFS
Web3
Distributed systems

ABSTRACT

This paper introduces the *IPFS requested content location service*, a software service to monitor the operation of IPFS from the perspective of the content requested through IPFS gateways. The software is provided as a docker stack that consumes the logs of one or more IPFS gateways, extracts the CID of the requested content and the IP address of the requester, and queries the IPFS network for the providers of the content. The software also matches the IP addresses of the requesters and providers with their geographic location, and stores the results in a database for later analysis. The software has been used in our previous measurement study, published at DAIS'23, that analyzed the operation of IPFS from the perspective of the content requested through gateways.

Code metadata

Code metadata description	
Current code version	v1.0.1.1
Permanent link to code/repository used for this code version	https://github.com/ScienceofComputerProgramming/SCICO-D-23-00391
Permanent link to Reproducible Capsule	Software runs with live data.
Legal Code License	GNU General Public License v3.0
Code versioning system used	git
Software code languages, tools, and services used	Go, Python, Docker, Nginx, Graphana, PostgreSQL
Compilation requirements, operating environments and dependencies	MaxMind GeoLite2 License Key, Linux, python3.10, libpq-dev
If available, link to developer documentation/manual	
Support email for questions	pah.costa@campus.fct.unl.pt

1. Motivation and significance

Decentralized large scale systems are becoming once again popular due to the popularity of blockchain technologies [1] and the emergence of the Web3 movement [2]. The InterPlanetary File System [3] (IPFS) is a peer-to-peer distributed file system that seeks to connect a large amount of computer devices to create a distributed and decentralized storage layer that can support the future web. IPFS is a content-addressable, peer-to-peer hypermedia distribution protocol, where nodes in the network publish (i.e., announce)

☆ This work was partially supported by FCT/MCTES grant SFRH/BD/144023/2019.

* Corresponding author.
E-mail address: pah.costa@campus.fct.unl.pt (P.Á. Costa).

https://doi.org/10.1016/j.scico.2024.103174
Received 15 December 2023; Received in revised form 10 June 2024; Accepted 11 July 2024

Available online 18 July 2024

and retrieve (i.e., search) content. Content is identified by a unique identifier called the *Content Identifier* (CID), which is a hash of the content itself. Nodes lookup content by querying the IPFS network for the CID of the content. IPFS leverages a Distributed Hash Table (DHT) protocol [4] to perform the efficient lookup of CIDs in the network. The DHT is maintained by the nodes of the network and stores *provider records* that contain the IP addresses of the nodes that store content. Users can access content in IPFS by running an IPFS node on their computer, or by using an IPFS *gateway* node that runs an IPFS node and exposes an HTTP API to retrieve content from IPFS, enabling the user to access the content via a web browser.

Although IPFS has grown significantly in the last years, there is still a lack of understanding of how IPFS operates. This is due to the decentralized nature of IPFS, where there is no central entity that controls the network and has access to all the information, which makes understanding and monitoring the operation of IPFS extremely daunting. Furthermore, the larger the network grows, further challenges arise in the efficient operation of the network, and is therefore imperative to develop methodologies and tools to retrieve information from the network about its operation.

This paper describes the software tool [5] that was developed to perform the work described in our DAIS'23 paper [6], that analyzed the operation of IPFS through the perspective of content requested through one of the most popular IPFS gateways. The software is a service that consumes the logs of one or more IPFS gateway nodes, extracts the CID of the requested content and the IP address of the requester, and queries the IPFS DHT for the provider records associated to the CID. The software further matches the IP addresses of the requesters and providers with their geographic location (by using the MaxMind GeoLite2 database [7]), and stores the results in a database for later analysis. This enables to study, from the perspective of IPFS gateway nodes, the frequency of requests for content, the amount of providers of (requested) content, and also the (approximate) geographic location of the requesters and providers of content.

Note that this software is not the first tool to be developed for monitoring and studying the operation of IPFS. In fact, IPFS (and other decentralized systems such as Ethereum [1]) have been the target of several studies that aim to understand how these systems operate. To achieve this, several tools have been developed to monitor and study the operation of these systems. We point to the Nebula crawler [8] and the IPFS CID Hoarder [9] as examples of tools that have been developed to study IPFS. Nebula is a crawler that crawls the IPFS DHT and collects related information about the membership of the network, such as the number of nodes in the network, the number of nodes per country, some network topology information, information about the versions of the IPFS nodes, and other related information. The IPFS CID Hoarder is a tool that measures the liveness of content in the IPFS network (i.e., how long does content remain discoverable in the IPFS DHT), by publishing random content in IPFS and then periodically measuring how many provider records for the published content can be found. The software described in this paper is complementary to these tools, as it analyzes the operation of the IPFS network from the perspective of the requesters of content (through the vantage point of IPFS gateways). As mentioned before, this allows to study the frequency of requests for content, the amount of providers of (requested) content, and also the (approximate) geographic location of the requesters and providers of content.

2. Software description

The software is a service that consumes the logs of one or more IPFS gateway nodes, extracts the CID of the requested content and the IP address of the requester, and queries the IPFS DHT for the provider records associated to the CID. The software further matches the IP addresses of the requesters and providers with their location (by using the MaxMind GeoLite2 database [7]), and stores the results in a database for later analysis. The software is provided as a set of docker containers that interact with each other to provide the service (i.e., a docker stack executed under a docker swarm environment). The software can be executed either in a continuous mode, where it consumes the gateway logs as they are generated as a stream, or in a batch mode, where it consumes the logs from a given time interval. In the following we detail the architecture and main components of this software.

2.1. Software architecture

Fig. 1 shows the software architecture. The software is composed by the following components:

- **Message Broker**: The message broker is responsible for receiving the logs from the IPFS gateway nodes and distributing log entry (i.e., line) to one or more Controller components. The message broker is implemented using RabbitMQ.
- **Controller**: The controller is responsible for consuming each log entry received from the message broker and coordinate the remaining components. This is the main component of the software and is implemented in Go.
- **Parser**: The parser is responsible for parsing logs into structured data (e.g., JSON), as well as matching the IP addresses of the requesters and providers with their location, by using the MaxMind GeoLite2 database [7]. The parser is implemented in Python and expects the logs to be formatted as described in [10].
- **Providers Finder**: The providers finder component is responsible for finding the providers of the content that was requested. This is done by querying the IPFS DHT for the CID, using the `libp2p` DHT library [11]. The providers finder component is implemented in Go.
- **Database**: The database is responsible for storing the data regarding the requesters and providers of content. The database is a PostgreSQL database whose database schema can be found in [10].

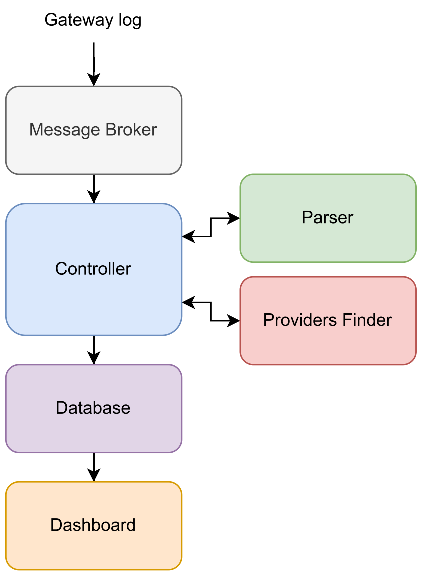


Fig. 1. Architecture of the IPFS requested content location service.

- **Dashboard**: The dashboard provides a web interface to visualize the data stored in the database. The dashboard is implemented using Grafana and Nginx. The Nginx server can be used as a reverse proxy to enable the dashboard to be accessed publicly.

2.2. Software requirements

As the software is containerized and provided as a docker stack the only requirement to run the software is to have Docker and Docker Compose installed. However, we also provide helper scripts (that can be found under the `scripts` directory in the repository) to interact with the software service. These scripts are implemented in Python and require Python 3.10 or higher to be executed. We provide a `requirements.txt` file (under the `scripts` directory) in the repository to install the required python dependencies. Furthermore, to interact with the Postgres database, the user also needs to have installed the `libpq-dev` library (on Ubuntu) or the equivalent library.

2.3. Usage

To execute the software, the user needs to have Docker and Docker Compose installed, Python 3.10 or higher, and be able to connect to the IPFS network (a simple Go program to test this is provided in the repository, that is also prepared to be containarized). Note that the software was tested on Debian 10, Ubuntu 22.04.1, and on MacOS 12.6.

Step 0. Clone the repository:

```
git clone \
https://github.com/pedroAkos/IPFS-location-requested-content.git
```

Step 1. Check if the user can connect to the IPFS network. To do this first build the `test_ipfs_connection` Docker image by running the following command in the root of the repository:

```
docker build -t test_ipfs_connection \
-f dockerfiles/test_ipfs_connection.dockerfile .
```

Then run the container:

```
docker run -it --rm test_ipfs_connection
```

If the container is able to connect to the IPFS network, it will print that the user should be able to connect to the IPFS network. If the container is unable to connect to the IPFS network, it will print that the user needs to troubleshoot the issue. Most commonly this is due to either firewall issues or DNS issues (to solve the latter, a good alternative is to use Cloudflare's DNS servers – 1.1.1.1).

Step 2. Request access to MaxMind database. To do this, go to https://www.maxmind.com/en/geolite2/signup and request access to the GeoLite2 database, which will provide the user with a license key.

Step 3. Build and run the Docker compose stack. To do this, run the following commands in the root of the repository:

```
export \
MAXMIND_LICENSE_KEY=<your license key>
docker-compose build
docker swarm init
docker stack deploy \
-c docker-compose.yaml ipfs-loc
```

Please check that all processes are running correctly by running the following command:

```
docker stack ps ipfs-loc
```

Some processes might be restarted several times due to dependencies among the containers, but all processes should eventually be running and stable.

2.4. Terminating and restarting the service

To terminate the service, the user needs to run the following command:

```
docker stack rm ipfs-loc
```

To restart the service, the user needs to run the following command:

```
docker stack deploy \
-c docker-compose.yaml ipfs-loc
```

To have a clean start (without any data in the database), the user needs to delete the database volume and restart the service:

```
docker volume rm ipfs-loc_db-data
docker stack deploy \
-c docker-compose.yaml ipfs-loc
```

3. Illustrative examples

In this section we present some illustrative examples of how to use this software. We use the helper scripts found in the repository's `scripts` directory. To follow the examples, the user needs to have the software service running (as described in the previous section), and have the python dependencies installed on their python environment of choice:

```
pip3 install -r scripts/requirements.txt
```

We provide a sample log file in the repository (`scripts/data/sample/sample_data.log`) that can be used to test the software. The log file contains the logs of an IPFS gateway node, and is formatted as described in [10]. The user can publish the log file into the message broker by running the following command:

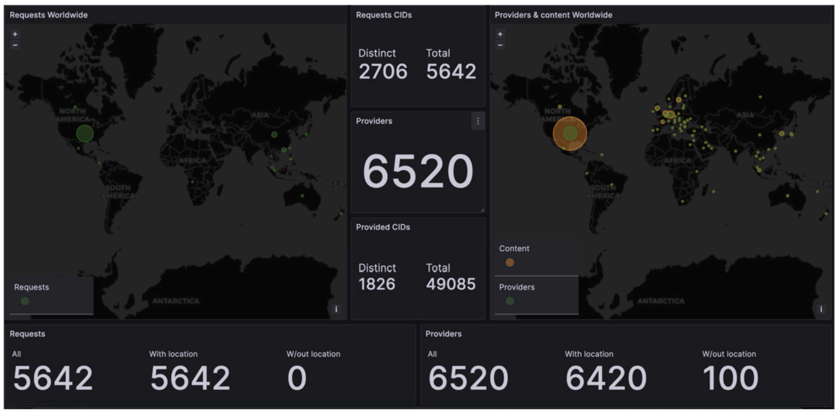
```
python3 -m scripts.publish_to_rabbitmq \
-t ipfs-gateway-logs \
< scripts/data/sample/sample_data.log
```

Note that the software service may take some time to fetch the information for each log entry. This means that the script will finish before the software service having gathered all the information from the IPFS network for each log entry.

Fig. 2 shows a part of the dashboard that is provided with the software, that can be accessed locally via the browser at http://localhost:3001 (the default username and password are both `admin`). Note however that the dashboard needs to be configured manually. To do this, the user needs to first add the data source (via the Graphana web interface). The data source should be configured as follows:

- Name: `PostgreSQL`
- Host: `db:5432`
- Database: `ipfs_content_location`
- User: `postgres`
- TLS/SSL Mode: `Disable`

After this, the user needs to import the dashboard configuration (again via the Graphana web interface) that is in the repository (the configuration file is: `dashboarb.json`), and select the previously created data source for the dashboard.


Requests Worldwide

Requests CIDs
Distinct 2706 Total 5642

Providers
6520

Provided CIDs
Distinct 1826 Total 49085

Providers & content Worldwide

Requests

Content

Providers

Requests
All 5642 With location 5642 W/out location 0

Providers
All 6520 With location 6420 W/out location 100

Fig. 2. Dashboard showing the amount of requests and providers per country.

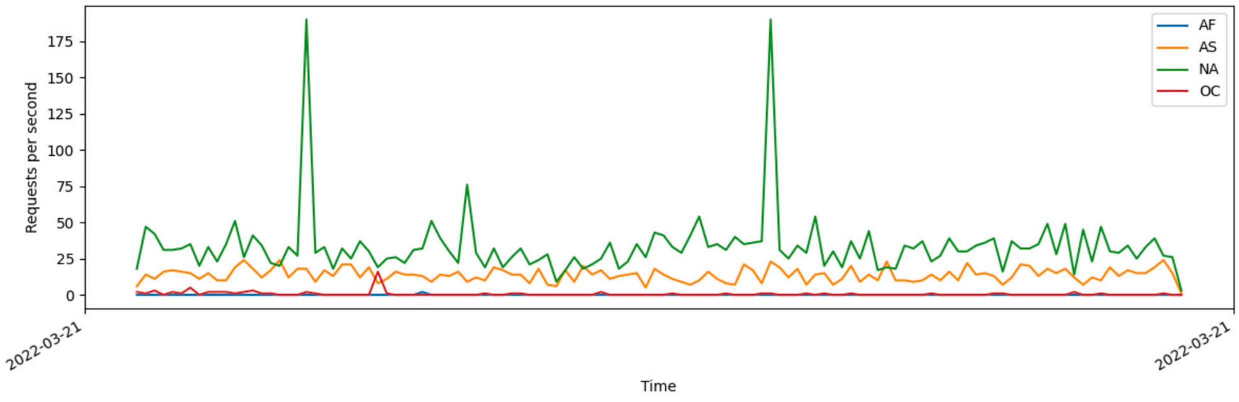


Fig. 3. Amount of requests per continent per second.

We also provide some scripts to analyze the data captured by the service, that were used to produce the results in our DAIS'23 paper [6]. Fig. 3 shows a figure produced by the analysis scripts that shows the amount of requests per continent per second. The analysis scripts can be run as follows:

```
python3 -m scripts.analyse_results \
--time-unit second
```

The script reads the database to retrieve the information gathered by the software service and saves the information in `csv` files in a defined data directory (by default named `data`). This is done to analyze large snapshot data without having to always query the database. The script produces various plots (similar to the ones presented in our DAIS'23 paper [6]) that are stored in a result's directory (by default named `res`). If the user wants to rerun the script with fresh data (as for this illustrative example the retrieval of the information from the IPFS network can take some time), the user should delete the data directory prior to re-executing the previous command.

4. Impact

As mentioned previously, the software was used in our previous measurement study, published at DAIS'23 [6], and further detailed in a report [10]. The software enabled us to analyze the requests made to IPFS via one of the most popular IPFS gateways, and collect the following information:

- The total amount of requests performed to the gateway.
- The total amount provider nodes found for the requested content.
- The amount of requests per geographic location (i.e., continent).
- The amount providers found per geographic location (i.e., continent).
- The correlation between the geographic location of the requester and the geographic location of the provider.
- And other related information.

5. Conclusions

The software described in this paper enables the monitoring of content requested through gateways in IPFS, and the analysis of the monitored data. The software is a service that consumes the logs of one or more IPFS gateway nodes, extracts the CID of the requested content and the IP address of the requester, and queries the IPFS DHT for the provider records associated to the CID. The software further matches the IP addresses of the requesters and providers with their location (by using the MaxMind GeoLite2 database [7]), and stores the results in a database for later analysis. The software was used in our previous measurement study [6] published at DAIS'23, as well as in an additional report that further details the study [10].

CRediT authorship contribution statement

Pedro Ákos Costa: Writing – review & editing, Writing – original draft, Software. **João Leitão:** Writing – review & editing, Supervision. **Yannis Psaras:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] G. Wood, Ethereum: a secure decentralised generalised transaction ledger, Tech. rep., arXiv:1011.1669v3, 2014.
[2] G. Korpal, D. Scott, Decentralization and web3 technologies, Tech. rep., https://doi.org/10.36227/techrxiv.19727734.v1, 5 2022.
[3] D. Trautwein, A. Raman, G. Tyson, I. Castro, W. Scott, M. Schubotz, B. Gipp, Y. Psaras, Design and evaluation of ipfs: a storage layer for the decentralized web, in: Proceedings of the ACM SIGCOMM 2022 Conference, SIGCOMM '22, 2022.
[4] P. Maymounkov, D. Mazieres, Kademlia: a peer-to-peer information system based on the xor metric, in: International Workshop on Peer-to-Peer Systems, Springer, 2002.
[5] P. Ákos Costa, pedroAkos/IPFS-location-requested-content: v1.0.1.1, https://doi.org/10.5281/zenodo.7850384, Apr. 2023.
[6] P.Á. Costa, J. Leitão, Y. Psaras, Studying the workload of a fully decentralized web3 system: Ipfs, in: M. Patiño-Martínez, J. Paulo (Eds.), Distributed Applications and Interoperable Systems, Springer Nature, Switzerland, Cham, 2023, pp. 20–36.
[7] MaxMind, Maxmind - geolite2 free geolocation data, https://dev.maxmind.com/geoip/geolite2-free-geolocation-data?lang=en, 2022. (Accessed October 2022).
[8] D. Trautwein, Nebula, https://github.com/dennis-tra/nebula, 2022. (Accessed December 2023).
[9] M. Cortes, Ipfs cid hoarder, https://github.com/cortze/ipfs-cid-hoarder, 2022. (Accessed December 2023).
[10] P. Ákos Costa, RFM3 | Location of IPFS end users and requested content, https://github.com/plprobelab/network-measurements/blob/master/results/rfm3-location-ipfs-users-and-requested-conten.md, 2022.
[11] Protocol Labs, libp2p: a modular network stack, https://libp2p.io, 2022. (Accessed February 2022).