Adaptive Edge Content Delivery Networks for Web-Scale File Systems

João Tiago
INESC-ID / Técnico Lisboa - ULisboa, Portugal

David Dias
Protocol Labs

Luís Veiga
INESC-ID / Técnico Lisboa - ULisboa, Portugal

Abstract—The InterPlanetary File System (IPFS) is an hypermedia distribution protocol, addressed by content and identities. It aims to make the web faster, safer, and more open. The JavaScript implementation of IPFS runs on the browser, benefiting from the mass adoption potential that it yields. Startrail takes advantage of the IPFS ecosystem and strives to further evolve it, making it more scalable and performant through the implementation of an adaptive network caching mechanism. Our solution aims to add resilience to IPFS and improve its overall scalability, by avoiding overloading the nodes providing highly popular content, particularly during flash-crowd-like conditions where popularity and demand grow suddenly. We add a novel crucial key component to enable an IPFS-based decentralized Content Distribution Network (CDN). Following a peer-to-peer architecture, it runs on a scalable, highly available network of untrusted nodes that distribute immutable authenticated objects which are cached progressively towards the sources of requests.

I. INTRODUCTION

The InterPlanetary File System (IPFS) [1] seeks to revert the historic trend of a client-server only Web. It is a decentralized peer-to-peer content-addressed distributed file system that aims to connect all computers offering the same file system. Due to its decentralized nature, IPFS is intrinsically scalable. As more nodes join the network and content demand increases, so does the resource supply. Such a system is inherently fault tolerant and, leveraging economies-of-scale, actually performs better as its size increases. It uses Merkle DAGs [2] [3], data structures to provide immutable, tamper-proof objects that are content addressed.

However, there are some crucial Content Delivery Network enabling features still lacking in IPFS. In particular, the system lacks the capability of swiftly and organically approximate content from the request path, reducing the latency incurred by future requests. It also does not prepare for the provider of an object to serve a sudden flood of requests, thus rendering content actually inaccessible to some/many.

The goal of this work is, by taking advantage of the ecosystem built by IPFS, to develop an extension to IPFS that implements an adaptive distributed cache at the edge to improve system’s performance, further evolving it. We aim to:

- reduce the overall latency felt by each peer;
- increase the peers’ throughput retrieving content;
- reduce the system’s overall bandwidth usage;
- improve the overall balance in serving popular content by peers;
- finally, improve nodes’ resilience to flash crowds.

In summary, Startrail serves as a key enabling component for an IPFS-based CDN.

II. ARCHITECTURE

We designed Startrail to be a functional edge-oriented network cache, one that continually moves content ever closer to a growing source of requests. The goal is reducing on average the time it takes to access content on the network. It does so without requiring intermediate nodes to previously request such content, thus enabling smaller providers to serve bigger crowds, this way addressing the cost and infrastructure barriers-to-entry of typical CDNs. It also aims to be interoperable, i.e. nodes running Startrail need not depend on other nodes to be effective, enabling nodes to contribute to the network even when adoption is not total.

Figure 1 depicts one of simplest scenarios with a small portion of a network where all the nodes are running Startrail: the content, in this case block $QmBlock1$, is stored on Node A. Nodes C and D request $QmBlock1$ to the network. While doing so, Node B that is requested by both, detects that the Content Identifier (CID) is popular and flags it, fetching and caching the content itself. Later, when Node E requests the block, the response needs not traverse the whole network, it can be served by Node B.

A. IPFS Core Architecture

Startrail is built on top of IPFS and integrates with some of its deeper internals and mechanics, thus we address its core. Objects on IPFS consist of Merkle DAGs of content-addressed immutable objects with links, with a construction
similar but more general than a Merkle tree [4]. Deduplicated, these do not need to be balanced, and non-leaf nodes may contain data. Since they are addressed by content, Merkle DAGs grant tamper proof, which is key to ensure the cached content is genuine. A high-level overview of the architecture of the IPFS core is depicted in Figure 2. We shall delve into each of the illustrated components:

- **Core API**: the Application Programing Interface (API) exposed by the core, imported by both the Command Line Interface (CLI) and the HTTP API;
- **Repo** - API responsible for abstracting the datastore or database technology (e.g. Memory, Disk, AWS S3). It aims to enable datastore-agnostic development, allowing datastores to be swapped seamlessly;
- **Block**: API used to manipulate raw IPFS blocks;
- **Files**: API used for interacting with the File System. textbfUnixFS is the engine for Unix files layout and chunking mechanisms for network file exchange;
- **Bitswap**: data trading module for IPFS. It manages requesting and sending blocks to and from other peers in the network. Bitswap has two main jobs: i) to acquire blocks requested by the client from the network; and ii) to send blocks it holds to peers who want them;
- **BlockService**: content-addressable store for blocks, providing an API for adding, deleting, and retrieving blocks. This service is supported by the Repo and Bitswap APIs.
- **libp2p**: networking stack and modularized library that grew out of IPFS. It bundles a suite of tools to support the development of large scale peer-to-peer systems. It implements typical P2P mechanisms, e.g. discovery, routing, transport through many specifications, protocols and libraries. One such library is **Kad-DHT**: the module responsible for implementing the Kademlia DHT [5] with the modifications proposed by S/Kademlia [6]. It has tools for peer discovery and content or peer routing.

Data exchanges on IPFS are handled by **Bitswap** [7], a BitTorrent inspired protocol. Peers operate two data-structures:

- **want_list**: set of blocks a node is looking to acquire;
- **have_list**: set of blocks a node offers in exchange.

When searching for a block, **Bitswap** will first search the local **BlockService** for it. If not found, it will resort to the content routing module, in our case, the **Kad-DHT** module. This will query the local providers database for the known providers of a certain CID and if not enough are gathered, query the rest from the DHT. Once obtained the set of potential providers, the node will connect to them and pass its **want_list** containing the target CID (illustrated in Figure 3).

**B. Startrail’s Extension Architecture**

We now proceed to describe how and where we integrate the Startrail cache as an IPFS extension to enable CDNs over IPFS. The first step is to identify where to tap into so that new content requests are intercepted. On IPFS, one can do that via two different ways:

- On **Kad-DHT**, checking for the CID associated to **GET_PROVIDER** messages;
- On **Bitswap**, listening for new **want_list** messages and tracking each of the included CIDs;

Our solution implements the former as it requires addressing just one type of messages.

Startrail’s main activity is to detect trends in object accesses. To do so, it uses two separate components, whose simplified class diagram is depicted in Figure 4:

- **Startrail Core**: exposing the Startrail API, it is the interface other components will consume to work with StarTrail. It is responsible for integrating with the data trading module (**Bitswap**), with the **BlockService** to access data storage, and libp2p for various network utilities.
- **Popularity Manager**: component responsible for tracking objects’ popularity. It is totally configurable and it can operate with any specified caching strategy.

The **Core** includes two main exposed functions: **process(cid)**, responsible for triggering the orchestration
and popularity calculation, returning a Boolean for ease of integration with Kad-DHT: updateConfigs(), used for configuration refresh, fetching new configurations from the IPFS Repo and if changes are detected, will update the live ones (useful for hot reloading configurations when testing).

In the Popularity Manager class we find: isPopular(cid), for updating and calculating the popularity for any CID passed as argument; updateConfigs(), serving the same purpose as the above mentioned one; nextTimeout(), managing the sampling timer responsible for scheduling timeouts; update(), runs every time the timeout pops, pushing the current sample to the sampling history and a new one is created.

1) Message processing algorithm: To recognize patterns in object accesses, Startrail examines the CIDs sent on discovery messages. Hence, the process() function is triggered every time the GET_PROVIDER message handler is called. Figure 5 depicts the execution flow starting when a peer requests a block from the network triggering the search for providers on the network. Upon receiving a GET_PROVIDER message, the Kad-DHT handler will execute Startrail’s process hook.

Following the popularity update, either no further action is required, or the block is flagged popular and the peer will attempt to fetch it or retrieve it from the BlockStorage. The block could potentially be found in the storage because, since we are using the IPFS BlockStorage, the block could have been previously fetched by either Startrail or the peer itself. Either way, subsequently to acquiring the block, the peer announces to the network that it is now providing it.

2) Popularity Calculation Algorithm: By studying the current and past popularity of a certain CID, we are able to likely forecast content that is going to, at least likely, remain popular in the future. Caching this locally and serving it to other peers has the benefit of making other nodes’ accesses faster. For simplicity, our forecast takes into consideration only a small subset of the node’s past. This subset, or window, can be obtained through various techniques. The one implemented in our solution is a hopping window. Here, sampling windows may overlap. This is desirable in our solution as we want to maintain some notion of continuity and smoothness between samples. Meaning that an object that was popular in the window before, still has high probability to remain popular in the current one, since a portion of the data remains the same. Although parameters are fully configurable, the defaults are 30 seconds for window duration, with hops of 10 seconds.

The Popularity Manager implements the hopping window by dividing it into hop-sized samples. In our case we divide the total 30 second sampling window into three 10 second samples. Every time a new message is processed, the Startrail Core checks the popularity of the referenced object by running the isPopular() function. The function will keep track of objects it has seen in the current 10 second window; incrementing a counter every time the CID processed. Every 10 seconds the current window, or sample expires and is pushed onto a list that holds the previous ones. It is on this latter list of samples (samples in the class diagram in Fig. 4) that the popularity calculations are made. An illustration of the interaction between samples and block arrivals is represented in Figure 6.

To calculate a block’s popularity the Popularity Manager will first select the three most recent samples after concatenating the current one to this list. Next, it will reduce the array outputting the total amount of times the object was observed. If bigger than a certain configurable threshold the object is considered popular. This heuristic has the benefits of (i) being fairly simple to implement and compute; it also (ii) reacts quickly to changes in content access trends. In our current configuration, it can be considered rather optimistic, since spotting the same object twice will consider it popular. Our heuristic does not take into account the size of the content being cached. This was a conscious decision, the reasoning for it is that on IPFS most blocks have the maximum default size of 256Kb. Usually only the last one of the sequence that makes up a file is smaller than that. Hence, we disregarded the block size as parameter for caching heuristic.

3) Cache Maintenance: In traditional caching systems [8], content is always cached by default with the cache replacement algorithm responsible for discarding less relevant documents to create space for new ones. Contrarily, Startrail employs an heuristic to judge which objects should be cached in first place.

We design Startrail so that, for the most part, it works
by leveraging the internal IPFS mechanics. This ensures the component is lightweight and uses the same procedures as the rest of the system. On Startrail, we allow the node to utilise the full amount of allocated storage by IPFS which defaults to 10 GB. Once it fills up, the IPFS garbage collector (GC) discards unnecessary objects. The IPFS’s GC removes the non-pinned objects. Hence, to prevent popular blocks from being collected when the it executes, we pin the popular objects.

When a block stops being popular it is unpinned, leaving it at the mercy of the GC. When the threshold of 90% of IPFS’ storage is reached blocks are no longer pinned in order to leave room for new blocks.

### III. RELATED WORK

Relevant work related to Startrail includes distributed storage and file systems (in particular peer-to-peer), content delivery networks, and web distributed technologies. Peer-to-Peer storage systems are distributed systems consisting of interconnected nodes (peers) serving content blocks. These can be: i) unstructured, when the network imposes no constraints on the links between different nodes. The placement of content is completely unrelated to the overlay topology, requiring little maintenance while peers enter and leave the system. These systems lack a way to index data (thus, resource lookup mechanisms consist of inefficient brute-force or gossiping), and limited scalability; or ii) structured, that fix the scalability issues of unstructured. Here, the overlay topology is tightly controlled and data is placed at specific logical locations in the overlay [9]. These systems provide a mapping between the data identifier and location, so that queries can be efficiently routed to the node(s) with the desired data, e.g. Chord [10], Pastry [11], Tapestry [12], and Kademia [5] that inspires the IPFS overlay.

**Content Delivery Networks** Content Delivery Networks (CDNs) are networks of geo-distributed machines that deliver web content to users based on their geographic location [13], including based on content naming [14] and leveraging wide area network topology [15]. Still, they improve content delivery speed and service availability only at the cost of owning or renting the geo-distributed replicas which are mostly fixed.

**Web Distributed Technologies** A few technologies are key for the successful design of decentralized systems in the Web. There are: i) the Browser, a very powerful tool to fuel adoption. Building a solution that runs on the browser means one is able to reach a broad audience; ii) Node.js allows running JavaScript in servers. It bundles with NPM (Node Package Manager), a very powerful library of reusable JavaScript modules; iii) WebRTC provides browsers and mobile applications with peer-to-peer Real-Time Communications capabilities, accessible through a Javascript API (with a suite of protocols addressing NAT traversal problems).

### IV. CONCLUSIONS

In this paper we presented a solution that extends the existing IPFS and improves it as a content sharing system and its ability to distribute it. After identifying the key architectural elements and functionality of IPFS and its shortcomings, we developed Startrail as an extension in the form of an adaptive caching component. The results (available in full in [16]) show that a network running Startrail nodes is able to perform better than one running only IPFS nodes. Startrail reduces the request latency by 30%, at the cost of small increase in total memory consumption of 20% while also reducing bandwidth utilization by around 25%. Additionally, we assessed the impact that different percentages of Startrail nodes have in the overall network performance. The results confirm the expectation that there is an inverse relation between Startrail nodes percentage and network latency. When one increases, then other is reduced.

**Acknowledgements** This work was supported by national funds through FCT, Fundação para a Ciência e a Tecnologia, under project UIDB/50021/2020.

**REFERENCES**

[1] Yiannis Psaras and David Dias. The interplanetary file system and the filecoin network. In 2020 50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume (DSN-S), pages 80–80, 2020.
[2] Pedro Teixeira, Hector Sanjuan, and Poyhtari Samuli. Merkle-CRDTs (DRAFT 1). pages 1–26, 2019.
[3] Merkle-dags. Technical report, Protocol Labs.
[4] Ralph C. Merkle. A digital signature based on a conventional encryption function. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 293 LNCS:369–378, 1988.
[5] Petar Maymounkov and David Mazières. Kademia: A peer-to-peer information system based on the XOR metric. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowston, editors, Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers, volume 2429 of Lecture Notes in Computer Science, pages 53–65. Springer, 2002.
[6] Ingmar Baumgart and Sebastian Mies. S/Kademlia: A practicable approach towards secure key-based routing. International Conference on Parallel and Distributed Systems - ICPADS, 2, 2007.
[7] Alfonso De la Rocha, David Dias, and Yiannis Psaras. Accelerating content routing with bitswap: A multi-path file transfer protocol in ipfs and filecoin. 2021.
[8] Waheed Ali, Siti Maryam Shamssuddin, and Abdul Samad Ismail. A survey of web caching and prefetching. International Journal of Advances in Soft Computing and its Applications, 3(1):18–44, 2011.
[9] Sameh El-ansary and Seif Haridi. An Overview of Structured P2P Overlay Networks. 2004.
[10] I Stoica, R Morris, D Karger, M F Kaashoek, and H Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. Sigcomm, pages 1–14, 2001.
[11] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, pages 329–350. Springer, 2001.
[12] Ben Yanbin Zhao, John Kubiakowitz, Anthony D Joseph, et al. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. 2001.
[13] Behrouz Zolfaghari, Gautam Srivastava, Swapnoneel Roy, Humid R. Nemati, Fatemeh Afgah, Takeshi Koshiha, Abolfazl Raz, Khodakhast Bibak, Pinaki Mitra, and Brijesh Kumar Rai. Content delivery networks: State of the art, trends, and future roadmap. ACM Comput. Surv., 53(2), apr 2020.
[14] Onur Aşciği, Vasileios Sourlas, Ioannis Psaras, and George Pavlou. A native content discovery mechanism for the information-centric networks. In Proceedings of the 4th ACM Conference on Information-Centric Networking, ICN ’17, page 145–155, New York, NY, USA, 2017. Association for Computing Machinery.
[15] Chavooosh Ghasemi, Hamed Yousefi, and Beichuan Zhang. Icdn: An ndn-based cdn. In Proceedings of the 7th ACM Conference on Information-Centric Networking, ICN ’20, page 99–105, New York, NY, USA, 2020. Association for Computing Machinery.
[16] João Tiago, David Dias, and Luís Veiga. Adaptive edge content delivery networks for web-scale file systems. arXiv 2207.06341, 2022.