Project Report

Alan Salimov

HS-Multihash, HS-IPLD Exploration

IPFS, or InterPlanetary File System, is a file system and protocol that acts in the same way as a large, single, bittorrent swarm. Files are scattered across users' local devices, with file sharing and access a product of following links across the entirety of IPFS. It is a robust, fault tolerant system with files existing in many places rather than one; it is permanent and distributed. Unlike HTTP, there are no large server farms to go down and render data inaccessible, and since access leads to ownership, data is not hidden from users, for better or for worse.

IPFS is currently implemented in Go and Node in varying degrees. My approach to the project was one of asking how to make things *better*, and how to start on that road as a newcomer to IPFS and distributed networks in general. This class offered a great opportunity to experiment with the advantages and disadvantages of a strongly typed, tragically functional language like Haskell in a huge system like IPFS with real world applications. Haskell brings security through typing and its function paradigm use means developing the skeletons needed for IPFS are easy and independent of the data involved. That said, at advanced levels

there are speed loses and the small Haskell community relative to the other implementations.

To understand how IPFS worked, I started from the ground level and took at look at the multi library. Multis are an effective way to describe addresses, hashes, codecs, and streams that retain salient metadata; I implemented a barebones multihash library that leveraged bytestrings to encode the type of hash and the size of the digest in a compact way. Everything in IPFS is addressed by these multi hashes; the nodes the system is build on being no exception. To ensure smooth handling these multi hashes need to have a common format and be decodable into a decoded multihash object and encodable back into a bytestring.

My implementation uses attoparsec to parse incoming hashes and to decode encoded multi hashes as well, returning Nothing in case the parse does not succeed. The type of hash is encoded in a standardized word8, with 0x11 reserved for sha1, for example. This format allows for a tremendous amount of flexibility and future proofing, since you can represent any hash.

The Merkle DAG (directed acyclic graph) is the heart of IPFS. Parent nodes have their childrens' nodes within, and you can follow a node in a quick and effective way all the way to the data layer itself. As a result, there's little distinction between a node and an edge, and edges can have data. Merkle DAGs aren't only for IPFS of course; it is a robust format that can be used to implement a filesystem, a torrent stream, etc.

To make a simple Merkle DAG, I set up a data object with a name hash record and then a nested "JSON" style map record. Looking up a file was a function of finding the name hash and then following the "directory" names, so lookup for "<hash>/a/b/cat.jpg" would descend into the map for a, b, and eventually access the cat.jpg file. That said, if it did not exist, then the links would be searched. Links' keys were "@link" and contained the hash of the children.

The hardest part was the nested dict format and having it typecheck correctly, but the immutability of data made it pleasant to make flexible functions that would work in a variety of use cases. The next steps for the Merkle DAG implementation after this is to be able to add new nodes and adapt it to a more IPFS specific interface. JSON input is also desired. As for this project as a whole is to implement IPLD completely and finish the multi libraries. Working in conjunction with others on Haskell IPFS and using the lessons learned to create an implementation that exposes flaws, fixes them, and capitalizes on successes in the existing IPFS go and node versions.