Kolmoblocks: composability for content distribution based on Merkle DAGs

Dmitry Borzov

(Dated: May 2018)

This paper proposes kolmoblocks (Kolmogorov data blocks): a Merkle DAG data block serialization format that enables block composability and self-documents the block uncompression procedure. Kolmoblocks can be thought of as scripts in a Turing-complete DSL that take other Merkle data blocks as an input and output the target data block. Its design assures reproducibility, code reuse and eliminates the issue of format versioning hell. Kolmoblock format can be used for content-addressed network protocols or any other system that relies on the Merkle DAG data structure.

I. INTRODUCTION

Content-addressable network protocols (or CANPs) such as IPFS[1] provide unique advantages over traditional host-addressable network protocols. The nodes that support content-addressable protocols share and seek data blocks based on their cryptographically secure hash of the content from any available peer node, not just on the specific host node that is expected to have it.

Current proposals for CANPs come with unique challenges:

1. **no composability**: in a scenario where the target blob is a slight iteration over the already distributed one - even when it's a single bit flip - would make a content-addressable network protocol treat it as completely new one; there is no capability to signal that a target block can be composed out of the other data blocks

2. **format versioning hell**: the context of the block's origination (what application version was used, the nature of data and so on) might determine the chosen block's serialization & compression format, it leads to to the unintentional consequence of most blocks being not comprehensible and useless after very little time.

Kolmogorov blocks are programs for a special DSL (the kolmoblock language) that output (or "render") the target block and can take other kolmoblocks as an input. Kolmoblock language interpreter specification makes kolmoblocks immutable, reproducable and Turing-complete.

Turing completeness means potential issue of the halt problem. Adding the concept of "gas", a limit on the computational resources that needed to render a kolmoblock, addresses this problem for trustless network.

This paper proposes kolmoblocks (or Kolmogorov data blocks): a block serialization format that addresses these issues. The design assures that kolmoblocks enable free-form block composability and self-documenting the block's format.

II. BACKGROUND

This section outlines the context behind the problem and the proposition value the kolmoblock design attempts to address.

A. Content-addressed network protocols

One can characterize all the modern network protocols as host-addressed.

For example, let's consider the task of accessing the text of Shakespeare's play "Much Ado About Nothing" from the internet. User's browser (the client) might send the following HTTP request to the IP address assigned to the IP assigned to shakespeare.mit.edu URL (the provider):

Cryptographically secure hash functions enable the family of content-addressed network protocols where the client requests content by its unique identifier. in contrast to provider-addressed protocols, the client can request and accept the requested content from any provider that supports the network protocol. Ability to verify the content's authenticity with the hash means the content can even be received from the trustless public networks.

Prominent examples of such content-addressed network protocols are IPFS [1] and dat [2]. As well pointed in the literature, the content-addressed protocols' comparative advantages make them a perfect fit for the following niche: content that tends to be bigger in size and that is distributed to multiple receivers [3].

For our example: the text of "Much Ado About Nothing" can be uniquely characterized by the following unique content identifier (serialized in git object format based on SHA-1 hash): 3f93390e9fd73eeebbee7b1220f56ea14230da3b.

The request to fetch the text within content-addressed protocol can look like

And could be addressed to any network peers that support the format.

B. Composability

Content-addressed hash functions are, by design, secure: it is considered to be practically impossible for two different data blocks to have the same hash.

Let's consider the scenario where the block's publisher (Shakespeare) edited the published text by adding one line to the beginning of the file: "A play by William

Shakespeare". That causes the content's identifier to completely change, and in order to get the client of the content-addressed network protocol will need to send the fetching request for the new version.

Let's consider the situation where the the client already has the copy of the old block. Instead of making the client download the new version as a completely new one, we could let the client know that it could "compute" the desired block by taking the original block and prepend with the line: "A play by William Shakespeare".

C. Compression/codecs

Let's stay we would like to improve the composability of network-addressed protocols.

For example, one may try extending the network-addressed protocol by supporting blocks composing insert operations. In our case the exchange between the client and the server:

III. DESIGN

Kolmoblocks (Kolmogorov blocks) are the programs for the Kolmogorov Virtual Machine, the output of which (the target blob) is the Merkle tree node for content-addressed protocols.

A client of a network addressed protocol that supports kolmoblocks can obtain a copy of the target blob by fetching the Kolmoblocks that advertise the desired block as the target.

Kolmoblock's header contains all the metadata that is necessary for the client to estimate the resources it will need to resolve the dependencies and evaluate the block.

A. Kolmoblock language

The Kolmogorov code is executed on Kolmogorov Virtual Machine (KVM).

Each kolmoblock specifies the maximum memory cap and the maximum number of KVM ops it takes for it to be rendered. KVM is Turing complete and has the following classes of ops: the classic boolean binary operations logical conditions and random gotos (which enables loops, Turing completeness and all that) random memory access confined by the block's memory cap KVM programs can only access the komoblock dependency blocks as the input; and the only output is the allocated target memory block. KVM is fully deterministic and reproducible as the design requires; that means KVM does not have any randomness-based or rendering agent-dependant behaviour

B. Clients and providers

CANPs nodes adopt the following practices to enable the support of kolmoblocks.

Let's say a CANP client is interested in fetching a block. It announces its interest to the providers and in return expects a list of kolmoblocks they can serve that has the block of interest as an output. Each kolmoblock in the list also has the other meta info such as the dependency blocks and gas, the metric of how computationally expensive it is to render this block. The client can make an estimate of the cost of obtaining and rendering each of the offered komoblocks with the following algorithm:

```
def evaluate_rendering_cost_of(kolmoblock):
    total_gas = 0
    total_bandwitch = 0
    if is_rendered(kolmoblock):
          return total_gas, total_bandwidth
    total_gas += kolmoblock.gas
    if not is_fetched(kolmoblock):
          total_bandwitch += kolmoblock.size

    for dep in kolmoblock.dependencies:
          dep_gas, dep_band = evaluate_rendering_cost_of(kolmoblock)
          total_gas += dep_gas
          dep_band += dep_band
    return total_gas, total_bandwitch
```

And then optimize to fetch and render the kolmoblock that would take the least of resources to render.

IV. DISCUSSION

In this section, we discuss the properties of kolmoblocks.

A. Self-extracting packages

Challenges of versioning, and serialization format versioning specifically, has been and will be one of the timeless companions of human condition. The situation where the used formats evolve so fast that people, and tools they rely on, are having trouble keeping up is sometimes referred as format versioning hell. The natural desire to address this issue in a systematic way is just as old, and the self-documenting serialization formats have been proposed and adopted throughout the history of technology.

While kolmoblocks are definitely is a continuation of this tradition, the context of CANPs and Merkle blocks provides some unique properties to it.

B. Global address space

Conventional programs has to choose between relying on the shared libraries provided by the host platform and statically including all the potentially usable code. The trade-off cons of using platform-provided code is that due to the versioning issues a different version of shared code could be used, which could lead to unspecified and non-reproducible behaviour of the program. On the other end, statistically including all the potentially used code leads to program code binaries ballooning in size with the bulk of code.

For example, a Docker image for Ubuntu starts at 188 Mb

In a way, all the kolmogorov code share global address space and can verifiably reuse all the code. The compositibility means the format specification can be isolated to a dedicated "format" block and then reused as many times as possible.

[1] IPFS specification, https://github.com/ipfs/specs
[2] dat protocol (https://www.datprotocol.com/)
[3] Juan Benet's comment on content-addressed protocols (https://github.com/ipfs/faq/issues/119#issuecomment-218278390)