The model of network map and data placement in the distributed decentralized storage platform

Anatoly Bogatyrev, Sergei Liubich, Fabian Wahle, Stanislav Bogatyrev, and Alexey Vanin

JSC "NEO Saint Petersburg Competence Center", St. Petersburg, Russia
{info,anatoly,sergei,fabian,stanislav,alexey}@nspcc.ru
https://nspcc.ru

Abstract. Nowadays, building an efficient, reliable and scalable decentralized data storage architecture is an actual problem in both corporate and academic communities since decentralization requires new approaches that adapt existing models of distributed systems. The aim of this research is to form a model for the decentralized storage system to distribute data over available storage nodes, efficiently reorganize data when nodes are added or removed and enforce flexible constraints on object replica placement that maximize data safety in the presence of coincident or correlated failures.
The model of distributed decentralized object storage integrated with blockchain payment system is proposed in this paper. A novel approach based on representing a network map as a graph and creating extended filtering mechanism for deterministic subgraph definition for object placement is introduced. The concept of a container, defining a network map subgraph, and a placement function for obtaining a container subgraph (based on the network map graph) or an object placement subgraph (based on the container subgraph) are presented. A modification of the CRUSH method is proposed for data placement.

Keywords: Network map · Placement function · Placement rule · Storage policy · Distributed decentralized storage platform.

1 Introduction

The development of blockchain technology has recently moved not so much towards global public permissionless blockchains as toward the implementation of corporate- and state use to solve specific internal tasks. It is logical to expect the development of Dapp projects in this direction. At the same time, DApps are supposed to store data in the decentralized storage systems to preserve their advantages over classic applications.

Currently, most projects [1, 2] in this field are aimed at implementing simple exclusive storages of data of some users on the capacities of other users or at creating add-ons via IPFS [3] for implementing public content distribution based on traditional hosting and CDN. In this case, the niche of storages that have a convenient API for DApp and the ability to organize isolated areas with control over data exchange in both public- and other private data storage areas is practically empty.

Building an efficient, reliable and scalable decentralized data storage architecture is an actual problem in both corporate and academic communities since decentralization requires new approaches that adapt existing models of distributed systems.

The aim of this research is to propose a model for the decentralized fault-tolerant [4] object storage system to distribute data over available storage nodes (a modification of the CRUSH [5] method is proposed for data placement), efficiently reorganize data when nodes are added or removed and enforce flexible constraints on object replica placement that maximize data safety in the presence of coincident or correlated failures.

The proposed model contains the following components: network structure, network map, data placement method and the concept of a container. These components are discussed in this paper.

2 The Model

2.1 Network Structure

The Distributed Decentralized Storage Platform (DDSP) p2p network is comprised of two types of nodes: Inner Ring nodes are responsible for maintaining information about network topology, accounting and data audit; Outer Ring nodes are responsible for storing data and ensuring its integrity and availability. Information about all active Outer Ring storage nodes and their properties forms a Network Map. Inner Ring is responsible for keeping Network Map up to date and distributes changes over all network peers. At the same time, Inner Ring nodes have to maintain all operations in the conditions of total distrust [7] of all network nodes. To solve this, it is proposed to use dBFT consensus protocol [8, 9].

2.2 Network Map

DDSP keeps the network map up to date and distributes it over nodes. It contains information about groups of nodes, their location in the network and main parameters necessary for correct data search and placement. The network map is represented as a graph [6] in the proposed method. The graph consists of vertices: buckets and nodes. A bucket is a vertex of the graph [5]. Together with outbound edges, it forms a subgraph of the network map, leaves of which are represented by nodes. Nodes can only be leaves, the bucket can be a parent for other buckets or nodes. The bucket is characterized by type and value (a key-value pair).

2.3 Placement Function and Placement Rule

Instead of maintaining and querying database with information about each object's location, the proposed DDSP uses consistent placement function with the following arguments:

- storage Policy defined with Placement Rules,
- network map (or the network map subgraph),
- rendezvous hashing salt.

The one can define a storage policy that is applied to the stored object. The placement rules consist of a set of SELECT() or FILTER() operations applied to the network map. The result of these operations is a subgraph of the network map where data can be placed. The SELECT() operation is applied to a tree-like subset of Network Map. It takes as inputs a replication factor and a bucket type. Multiple operations in the storage policy are enqueued and each subsequent operation is recursively applied to the previous operation's output. The FILTER() operation is applied to the graph. The operation inputs are the bucket type, bucket value and comparison operation. For text values, the operations eq and ne are available. For numeric values, gt, ge, lt, le are additionally available.

A set of operations on the graph (in the placement rule) can be grouped by using AND, OR, NOT. The placement function is performed recursively, with the operation of the next step being performed for all the nodes retrieved at the previous step. Buckets and placement nodes in the bucket are selected using Rendezvous hashing algorithm. With it, each node or bucket has an individual hash number for an individual item, and a bucket or node having the largest hash number are chosen. Data in this algorithm is distributed uniformly and small number of data movement is required when nodes are added or removed [5]. However, it can achieve minimum data movement when nodes are added or removed.

The placement function is executed recursively with the operation of the next step being applied for all the nodes retrieved at the previous step.

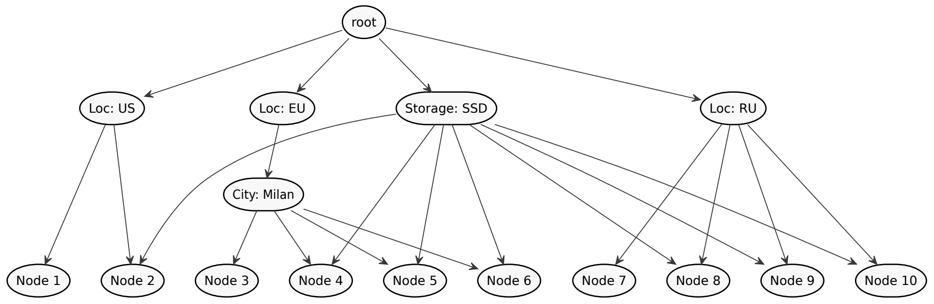

Fig. 1. Network map example.

This network map (or network map subgarph) (see Fig. 1) consists of the bucket with types Loc (country), City (city), Storage (storage disk type) and the nodes of the corresponding buckets. The storage policy may be defined as follows:

- it is stored in 2 different countries,
- it has 3 copies per each country,
- it should be stored on SSD drives only.

This informal description should be represented as sets of SELECT(r, type) and FILTER(type:value, op) operations. The total replication factor is obtained by multiplying all the factors of each SELECT(r, type) replication. The final statement must always be SELECT(X, Node) (see Table 1).

Table 1. The placement rule application to the network map graph.

Placement rule	Bucket result	Placement group result
SELECT(2, Loc)	[Eu] ∪ [Ru]	[[Node 3, Node 4, Node 5, Node 6], [Node 7, Node 8, Node 9, Node 10]]
FILTER (Storage:SSD, equal)	([Eu] ∩ [Storage:SSD]) ∪ ([Ru] ∩ [Storage:SSD])	[[Node 4, Node 5, Node 6], [Node 7, Node 8, Node 9, Node 10]]
SELECT(3, Node)	($[N_1, N_2, N_3] \in$ ([EU] ∩ [Storage:SSD])) ∪ ($[N_4, N_5, N_6] \in$ ([Ru] ∩ [Storage:SSD]))	[[Node 4, Node 5, Node 6], [Node 8, Node 9, Node 10]]

Thus, the subgraph of the network map and the 6 nodes have been obtained (see Fig. 2), where the object should be placed on to meet the placement policy requirements. The result of the placement function operation is a subgraph of the network map – the Placement group – leaves of which are a deterministic and consistent list of the storage nodes. If the storage policy, salt and network map are known the Placement group can be retrieved without referring to a third party or storing "object-storage node" pairs.

2.4 Container

In order to place objects in the system a user needs to define a container. The container has the following fields: storage policy (placement rule and redundancy factor), uuid, owner and maximum capacity. The container defines the subgraph of the network map. The approach, where a complete network map, which operates with all the connected nodes, plays the role of an initial graph, can impose additional costs as the network increases. The allocation of limited space within the network map (subgraph), defined by the container, allows to do the following:

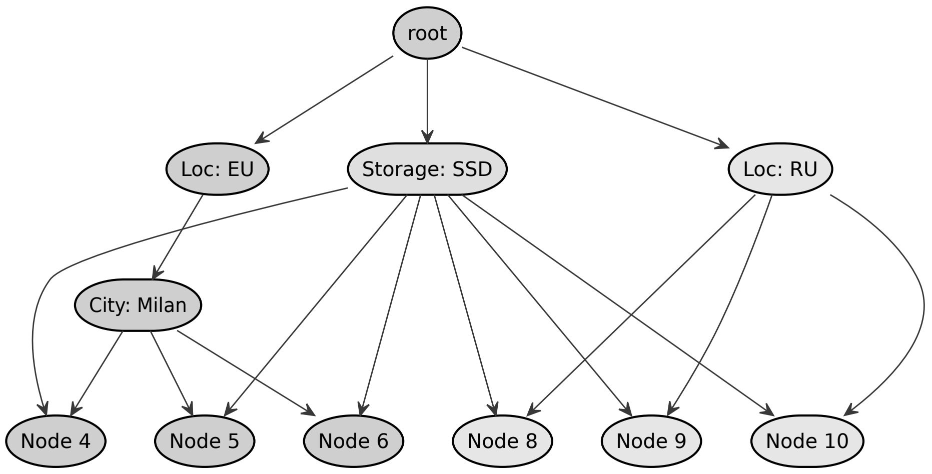

Fig. 2. The obtained subgraph of the network.

- isolate a subset of nodes. Handling of the subset is faster and more predictive. After the container has been provided with a graph, container traversal operations are faster,
- protect against overfilling – the overall container‘s capacity and its free capacity can be roughly estimated.
- Control access to the container,
- simplify payment operations by linking them to the container,
- facilitate the integration of s3 and swift API for DDSP.

To form the container‘s subgraph, the replication factor in each SELECT (r, type) call is increased by the redundancy factor (K_r). The container uuid is used as salt for selection in the placement function.

For ease of presentation, consider the example, where $K_r = 2$ see Table 2.

Table 2. Container.

Source	Placement rule	Placement function result	Salt
Network Map	SELECT($2 \cdot K_r$, Loc)	[US , EU, RU, Ja]	Container salt
	SELECT($2 \cdot K_r$, node)	$[[N_1, N_2, N_3, N_4]$, $[N_5, N_6, N_7, N_8]$, $[N_9, N_{10}, N_{11}, N_{12}]$, $[N_{13}, N_{14}, N_{15}, N_{16}]]$	

As a result, a number of nodes is obtained that can be represented as a subgraph of the network map. When forming the container, it is possible to approximately estimate its capacity. To do so, the weights of the nodes might be used, depending on their capacity. The weight of the selected nodes has to

Table 3. Placement group.

Source	Placement rule	Placement function result	Salt
Container	SELECT(2, Loc)	[US, Ja]	Hash of the object being placed
	SELECT(2, node)	[[N_2, N_4], [N_{15}, N_{16}]]	

correspond to the container capacity declared. The object is placed into the container by using the placement function, where the hash of the object being placed is used as salt (see Table 3).

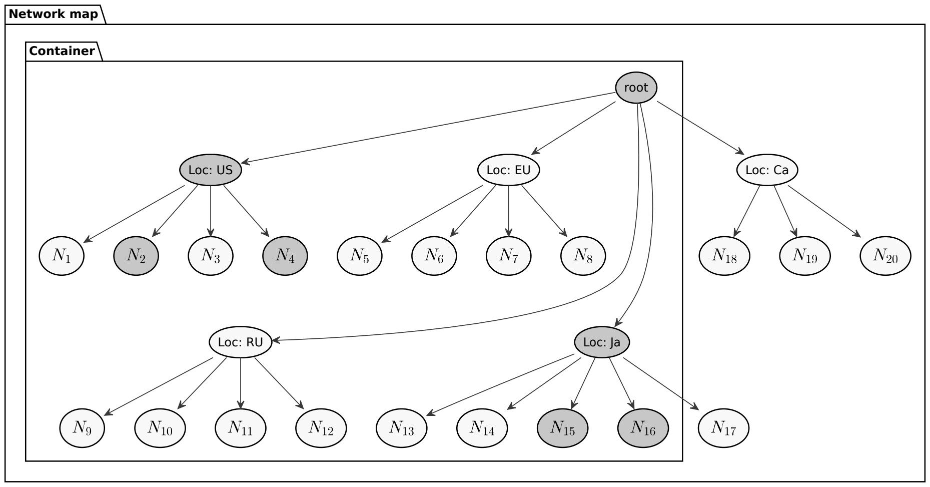

Fig. 3. An example of the placement function's result.

The highlighted buckets are the Placement group subgraph for the object. (see Fig. 3).

3 Results

The paper proposes a novel approach to scalable data placement in decentralized distributed storage systems. Using a subset of network map, storage policy rules for object placement, and a Rendezvous hashing for a node selection allow to control an object location and a minimal data movement in case of storage nodes failures. Metrics comparison with the existing models are the subjects of further research.

References

1. FileCoin Homepage, https://filecoin.io/. Last accessed 30 Jan 2019

2. NKN Homepage, https://www.nkn.org/. Last accessed 30 Jan 2019
3. "IPFS - content addressed, versioned, p2p file system", https://github.com/ipfs/ipfs/blob/master/papers/ipfs-cap2pfs/ipfs-p2p-file-system.pdf. Last accessed 30 Jan 2019
4. Bogatyrev, V.A.: Exchange of Duplicated Computing Complexes in Fault tolerant Systems. In: Automatic Control and Computer Sciences, vol. 45, no. 5, pp. 268-276 (2011)
5. Weil, S.A., Brandt, S.A., Miller, E.L., Maltzahn, C.: CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data. In: ACM IEEE SC 2006 Conference (SC'06), (2006)
6. Weil, S. A., Brandt, S. A., Miller, E. L., Long, D. D. E., Maltzahn, C.: Ceph: A scalable, highperformance distributed file system. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI), (2006)
7. Kallahalla, M., Riedel, E., Swaminathan, R., Wang, Q., Fu, K.: Plutus: Scalable Secure File Sharing on Untrusted Storage. In Proceedings of the 2Nd USENIX Conference on File and Storage Technologies, FAST '03, pages 29–42, Berkeley, CA, USA, 2003. USENIX Association, (2003)
8. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The Honey Badger of BFT Protocols. In Conference: the 2016 ACM SIGSAC Conference, (2016)
9. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In Proc. USENIX Annual Technical Conference, pp. 305–320, (2014)