2MEME

Wyatt L. Meldman-Floch

11/27/2023

Abstract

MEME, a reputation model for p2p systems based on peer metadata and cryptographically secure hashes, is presented. It is dynamic, not relying on a set list of trusted peers, rather selecting the most accessible peers as leaders: those with the minimal entropy rate relative to all peers, or peers producing the most correct information. The core application and focus is optimizing consistency of a multi-layered consensus protocol.

Contents

Introduction

For a distributed system to maintain consistency (as in CAP theorem), it needs to optimize information gain and minimize discrepancies. One set of approaches rely on trust or reputation models to select between potentially conflicting updates from untrusted peers. There are many approaches to solving reputation problems in p2p networks. The most famous is Eigentrust, and many expansions upon the base framework, such as Honestpeer and Powertrust. Due to the curse of dimensionality they all employ some type of random walk to explore the search space of transitive trust between nodes, calculating a probability distribution via monte carlo integration over probabilistic scores of all peers, provided by each peer. These expansions typically focus on finding new features or representations of trust, such as in deepwalk or node2vec, which create an embedding of social data to normalize the edge weights of the peer graph. This paper follows a similar approach using entropy or disorder across peer

behavior and is specifically applied to a dag-based multi-layered consensus protocol. Whereas many approaches such as Eigentrust require a seed or whitelist of authority nodes to base trust upon, this is insufficient when requiring decentralization such as for distributed consensus networks like cryptocurrencies. MEME circumvents this by determining correctness without pre-trusted peers, allowing nodes to join and leave and preventing centralized control over consensus. The core of the algorithm extends from the principle of maximal entropy, however applied this in reverse. The maximum entropy principle states that new information added to a system increases disorder relative to the previous state of that system, increasing proportional to novel information added. However, in the system architecture (described below) the data structures themselves optimize the partitioning of incoming data via rumor based gossip such that discrepancies between peer state form cliques representing potential network partitions. Periodically, a self avoiding random walk is performed on a graph of nodes as peers and edges as a vector of the entropy rate between data processed by each peer. The model chooses correct nodes by node influence metric based on node availability, which is defined as the most strongly linked nodes; ranking the peers/nodes by how similar their proposals are as opposed to how diverse the data is. The goal of MEME is to improve upon PRO models by explicitly using information gain metrics to converge on an accepted state of a distributed system, via consensus.This achieves real elasticity comparable to elastic infrastructure like Elasticsearch and Elastic Map Reduce, as well as objective decentralization operating without subjective human input.

System Architecture

The system considered here consists of a two layer consensus protocol, with two separate consensus processes, L1 and L0, directly influencing each other. Future work incorporating more consensus layers can be formulated using the poincare protocol and protocol topology specified in the author's previous work, Blockchain Cohomology. L1 peers perform a federated consensus, converging on the state of each peer's state cache. The contents of each state cache is a ledger of Addresses and collection of Transactions: data structures performing the transfer of a numerical amount (tokens) from one Address to another Address. Each Address has an associated linked list formed out of Transactions sent from this Address. The links are recursive cryptographic signature hashes between each sequential transaction at discrete Ordinal. Each 'owner' peer selects two 'facilitator' peers to share its state cache with. The two facilitator peers also share their state cache with each other and then the owner peer. The output of this process is a data structure, signed by the owner and facilitators, called a 'block' which consists of each peer's state cache data and two 'parent edges' called Tips, which are hashes of previous blocks. This can be conceptualized as a 'triangulation of state' which forms a forward arrow of

time out of parallel-process state transition data. This is realized as a data structure called the 'Data Dependency Graph' which is a directed acyclic graph of blocks with two parent edges, and three dimensions; namely, height, width and depth. The Data Dependency Graph as well as the ledger of Addresses has a poset topology, from which the forward arrow of time can be constructed out of parallel events. Each block is then sent across the L1 and L0 peers via rumor based gossip, so that they can be used as 'Tips' to form edges between old and new blocks. Tips and facilitators are selected using a pure function that seeks to maximize the area (in terms of height, width and depth,) this increasing the potential parallelism by enforcing consistency across all peers. L1 nodes are pruned according to an entropy calculation by L0 nodes on their blocks created; each block is created deterministically according to the total set of tips and peers, deviations from this result in low trust (and rewards). L1 trust scores are then normalized according to poisson distribution with low trust outliers being removed and addresses temporarily blacklisted (should we freeze funds too like slasher?) L0 peers perform a distributed consensus, $O(\text{peers}^2) data sent across these set of L0 peers, converging on the state of each peer's state cache. The contents of each l$
$diff interval, the L0 nodes cycle between 'active' and 'passive' states. The total number of active L0 nodes fluctuates$

Minimizing the Entropy Rate

Information gain can be formulated as the reduction of entropy or disorder in a dynamical system and depending on the characteristics of the system, it is calculated in one of many ways. For the purposes of MEME, which is formulated for application to consensus networks, it is calculated as a stochastic process. A stochastic process is an indexed sequence of random variables that do not need to be independent or identically distributed. In a consensus network, each peer continuously proposes variable state data, converging on an accepted state according to the rules of the consensus algorithm. This state data, in our case called blocks, can be independent or dependent on each other; and the amount of blocks as well as the specific blocks proposed can fluctuate or differ completely. Each block has an indexed order, or in the case of the system architecture above, a poset topology; meaning that they are strictly ordered. Thus these distributed systems fulfill the requirements of a stochastic process and can be modeled as such. While it is possible to apply MEME to linear blockchain protocols, it was formulated specifically for use in the system architecture above, with three indices: height, width and depth. The following formulas are specific to this poset topology. Consider a set of peers $N = (n_0 \dots n_i), acting as random variables which produce outputs O = (o_{0,0,0} \dots o_{h,w,d}), such that h < w and w > d, the system has a strictor der given by poset topology. These indices can be N, the entropy > 0. Conversely, if nodes propose blocks such that their indices conflict with other proposals, they co$

permissionless vs permissioned approaches

Two algorithms for calculating entropy rate are presented below. The key difference is the rate and method of calculating entropy rate. The first enforces a service level agreement requiring each peer to train its model at the same rate, achieving greater determinism and enabling a token reward model for an open network. In the batch model, at every snapshot height-diff interval, each peer proposes a new predicted trust vector within their proposals. They are then used to weight peer proposals for majority snapshot calculation until snapshot height-diff interval +1. The second, online algorithm, is a greedy approach that reduces the in-memory cost of running each peer's model on a deterministic schedule albeit at the loss of determinism that would allow a fair token reward model. It is more suited perhaps to applications that can relax determinism for open network rewards to focus on elasticity. The online algorithm periodically gossips predicted trust vectors to peers over the Peer api endpoint (" /trust"), which are then cached and fed into the TrustManager on a time based periodic interval.

TODO: type set below using pseudocode format

Batch Algorithm: Entropy Rate (Optimal for node rewards/cycling between active and passive nodes) Fix a discrete even number representing an index range called 'cycle' As each proposal of blocks is received, each node calculates the greatest common subset of blocks for each node, update cache of [blockhash: nodeId] At index number cycle, run monte carlo simulator At index number cycle*2, propose monte carlo simulation outcome, clear cache Repeat 2-4

Online Algorithm: Approximate Entropy Rate (current implementation, optimal for minimal resource usage, training model over shorter periods should help output, spamming results/sybil collusion should be detected by model, good test)

Fix time interval on nodes As each proposal of blocks is received, each node calculates the greatest common subset of blocks for each node, update cache of [blockhash: nodeId] Once time interval ends, pass cache of GCS to monte carlo simulator At next time interval end, propose monte carlo simulation outcome, clear cache Repeat 2-4

Monte carlo simulation: estimation via self avoiding random walk

(Note, to avoid confusion between a consensus network and graph, nodes are called servers below.) Next a self-avoiding walk is employed to perform commu-

nity detection, the output of which can be used to calculate availability and node influence. The output of the entropy rate calculation is a graph, with a server's peers corresponding to nodes and edges as the relative joint entropy between the server and its peers. The edges are a 'view' of the performance of each peer relative to itself. This is passed to the TrustManager, a background process that performs a self-avoiding random walk across the graph of nodes connected by relative joint entropy and outputs a vector containing a trust score for each node relative to the server hosting running the process (predicted trust). The self avoiding walk is performed by the method runwalkfeedbacksinglenode, which performs a series of feedback rounds, walking on the input graph and adjusting the edge weights between nodes for each successive round. The total number of feedback cycles are configurable and in general the larger number of cycles has a more accurate output, albeit at the cost of increasing resource intensity. The configurations are batchIterationSize and maxIterations. On each batch iteration a random path length from a random number generator (between 1 and total nodes) is chosen and then passed to the walk. The walk goes through and only walks on positive edges (relative entropy scores are normalized between -1 and 1), keeps track of nodes visited so far, then the sampling function determines the next neighbor to walk on. This is determined according to the normalized probability (via method normalizedPositiveEdges), such that the positive edge subset sum up to one. As this iterates, transitive trust scores are added, because products of trust are quite small. However, over many iterations they sum, to large numbers which is better for differentiating between scores. The main walk function walk() gets invoked by walkFromOrigin() inside runWalkRaw() which iterates over numIterations, adding up the scores into val walkScores for each node, removing the server's own. After that function is called, one "batch" has been created. Finally there is batch convergence in runWalkBatchesFeedback. This converges when a delta variable, which is just root mean squared error, becomes less than or equal to an epsilon variable, where epsilon is set to 1e-6; i.e the function terminates when scores don't change between batches by one part in a million. The output of the walk, batchScores, is not normalized, so they are renormalized by the Normalize function between each iteration until convergence. One difference from similar models is the incorporation of negative scores. After the batches are performed, it explores the negative scores (val negativeScores) of nodes that it trusts (positive outputs of the walk). On the first cycle, the model only reaches nodes with positive transitive trust, which can be considered the most influential servers. These most influential servers are relative to the host server, and the servers they distrust have their scores down weighted. The positive scores and negative scores are added, then weighted by how influential the server proposing these scores is. They are weighted such that its negative edge trust quantity*(influential node's score/numNegEdges), after that they are all normalized via renormalizeAfterNegative. Output is $\mathrm{P}_i^h below$

Experimental results

TODO put images from simulation here

Further investigation

Modifications to the self-avoiding walk implementation could yield positive effects. As in many monte carlo integrations, the direction chosen at each step could be chosen according to a distance metric as opposed to randomly. Albeit, at the computational cost of increasing the dimensionality the trust graph's edges as well as in-memory expense of the metric calculation. Notably, this approach was employed by (N. Koroviako) who used jaccard similarity to define trust out of relationships between review texts; as MEME approach focuses on information gain, it would follow to select the next path at each step based on minimizing entropy rate of second order proposals (each edge would contain raw proposals, and choose the next node that has the minimal entropy rate compared to the current's proposals.) Use Hausdorff clustering to identify/visualize hierarchies. Could have applications in further anomaly detection extending to higher dimensions of entropy in tandem with graph embedding or deep learning approach.

References

[1] Baez, John C *Circuitry, EE and chain complexes.*
`https://math.ucr.edu/home/baez/week288.html`

[2] Baez, John C *Physics, Topology, Logic and Computation:A Rosetta Stone.*
`https://arxiv.org/pdf/0903.0340.pdf`

[3] PIETER HOFSTRA AND PHILIP SCOTT *ASPECTS OF CATEGORICAL RECURSION THEORY.*
`https://arxiv.org/pdf/2001.05778.pdf`

[4] David Spivak, Brendan Fong *Seven Sketches in Compositionality.*
`https://math.mit.edu/ dspivak/teaching/sp18/7Sketches.pdf`

[5] Pierre Baudot *The Poincaré-Shannon Machine: Statistical Physics and Machine Learning Aspects of Information Cohomology.*
`https://www.mdpi.com/1099-4300/21/9/881`

[6] Tobias Fritz *A synthetic approach to Markov kernels, conditionalindependence and theorems on sufficient statistics.*
`https://arxiv.org/pdf/1908.07021.pdf`

[7] Tatsuya Hagino *A Categorical Programming Language*.
web.sfc.keio.ac.jp/ hagino/thesis.pdf