catalyst

Catalyst Network

Technical White Paper

Catalyst Network - Technical White Paper

Joseph Kearney*- Atlas City

November 4, 2019

Abstract

hello

*joseph.kearney@atlascity.io

Contents

Introduction

1 Distributed File System

Catalyst integrates its own DFS based on the InterPlanetery File System (IPFS) protocol [1].

2 KVM

The Catalyst network

2.1 From the EVM

2.2 To the KVM

3 Catalyst Consensus Mechanism

On distributed networks there is no single point of trust to determine the validity of transactions, therefore concurrency must be ensured by other methods. Typically this requires a majority of the network's participants to agree on a particular update of the ledger and the account balances / holdings held within. Blockchain technologies generally employ Proof-of-Work (PoW) and occasionally Proof-of-Stake (PoS) mechanisms in order to gain consensus across a network. However, these methods are prone to increasing centralization at scale as well as in the case of PoW high energy consumption. Other networks employ a small amount of trusted nodes that ensure the validity of transactions, however this is highly centralised and almost as fallible as the single point of failure systems that DLT endeavors to avoid.

Catalyst integrates a newly designed consensus mechanism, based on Probabilistic Byzantine Fault Tolerance (PBFT). This is a collaborative rather than competitive protocol, meaning that all honest work performed by nodes on the network benefits the security of the network and that all participating nodes are rewarded equally. For each ledger cycle a random selection of worker nodes are selected, the nodes become the producers for a cycle or number of cycles. The producer nodes perform work in the form of compiling and validating transaction thereby extracting a ledger state change for that cycle.

The protocol is split into four distinct phases:

- Construction Phase - Producer nodes that have been selected create what they believe to be the correct update of the ledger. They then distribute this proposed ledger update in the form of it hash digest.
- Campaigning Phase - Producer nodes designate and declare what they believe to be the most popular ledger state update.
- Voting Phase -
- Synchronisation Phase - In this phase the producers who have computed the correct ledger update can broadcast this update the rest of the network.

3.1 Notation

3.2 Producer Node Selection

3.3 Construction Phase

The first phase of the Catalyst consensus algorithm is the Construction Phase. Within which the selected producer nodes $P_j \ \forall j \in P$ calculate their proposed ledger state update or their local ledger state update. This is done by aggregating and validating all transactions that have occurred during a set time period. These transactions assuming their validity are integrated into the producers local ledger state update. From which they can create a hash of the update. This hash digest represents what they believe to be the correct update and is broadcast to the other producer nodes during for that cycle. Assuming the collision free nature of hash functions, the only mechanism for multiple producer nodes to have the same local ledger state update is by both using the same set of transactions.

The first phase starts at $t = t_p = t_{n,0}$ and lasts for a period of time Δt_p, therefore ending at $t_p + \Delta t_p$.

3.3.1 Local partial ledger state update generation and broadcast

Each producer in the set of producers P follows the same protocol. The construction phase begins with producer P_i beginning their construction phase by flushing their mempool. This mempool is made up of $\{T_i\}_{i=1,..,n}$ [BE MORE CLEAR WITH THIS DEF] where n is the number of transactions that have been broadcast to the network and have been stored by P_i. These transaction are used to create P_i's local ledger state update U_i.

The producer at this point also creates a hash trees d_i, this is to store the the signatures that are extracted from each transaction in T_i. A salt σ is created utilizing a pseudo-random number generator using the previous ledger state update U_{n-1} as its seed. P_i then follows the following steps:

1. Producer P_i verifies that each transaction in $\{T_i\}_{i=1,..,n}$ is valid following the rules set out in [ADD CITATION FOR GITHUB WITH VALID TRANSACTION RULES]. From each of the transactions in $\{T_i\}_{i=1,..,n}$ the entries that constitute the transaction are extracted to form a list $\{E_\alpha\}_{\alpha=1,...,m}$. The producer should therefore end up with as many lists as there are valid transactions from $\{T_i\}_{i=1,..,n}$. each signature from the transactions are also extracted and added to d_i.
2. P_i for each $\{E_\alpha\}$ it created then creates a corresponding hash digest as:
$$O_\alpha[CHANGETHISVARIABLE] = \mathcal{H}[E_\alpha \mid\mid \sigma]$$
Each pair (E_α, O_α) is added to a list L_i.
3. P_i then sorts list L_i into lexicographical order according the hash values O_α.
4. The producer P_i then extracts the transaction fee value from each transaction in $\{Tx_i\}$ to create v_i which is the total sum of all transaction fees.
5. The local ledger state update U_i for producer P_i can then be calculated. Firstly the list L_i is concatenated with the hashtree d_i and a hash digest is created as:
$$U_i = \mathcal{H}(L_i \mid\mid d_i)$$
U_i is then concatenated with P_i's unique peer identifier Id_i to create:
$$h_i = U_i \mid\mid Id_i$$
6. h_i is then broadcast to the other producer nodes on the network.

3.3.2 Partial ledger state update collection

[THIS SECTION NEEDS IMPROVING]

Producer P_i also collects other producers partial ledger update values. At most they will collect $P - 1$ values. Optimally every prducer in P will recieve the same set of transactions therefore for every $P_i \in P_{1,..,j}$ will have the same partial ledger update U_j. However this is unlikely due to all transaction not being received by a small group of nodes. Equally they may not hold G_i where $\{G_i\}_{i=h_1,..,h_j}$, meaning they may not receive a proposed update from all candidates.

3.4 Campaigning Phase

The second phase of the consensus mechanism is where producer P_i designates a candidate for what it calculates to be the most popular ledger state update.

3.4.1 Local candidate generation and broadcast

Beginning this phase producer P_i has a set of partial ledger updates G_i that it has received from other producer nodes. Each h_j within G_i contains a producers hash of the proposed update (U_j and their peer identifier (Id_j). The most popular U_j value can be found, this gives us U_j^{maj} from there the subset G_{maj} can be created, which is the amount of votes for the most popular update. Two thresholds must be considered first is G_min this is the minimum amount of updates it has received from other producers in order to generate a valid candidate. The second is G_{thresh} this is the threshold value for which a minimum number of votes must be in favor of G_{maj} which the most popular vote found within G_i. So in order to proceed with declaring a candidate $G_i > G_{min}$ and $G_{maj} > G_{thresh}$.

If the thresholds are met the following can take place:

1. P_i creates a list $\mathcal{L}_i(prod)$. To this list P_i appends the identifier of any producer that correctly sent the U_j value that equals U_j^{maj}. If P_i's U_i value is also the same as U_j^{maj} then they should append their own Id_i.
2. Producer P_i then creates their candidate for the ledger update c_i which is calculated as $c_i = U_j^{maj} \mid\mid \#(\mathcal{L}_i(prod)) \mid\mid Id_i$
3. Producer P_i will then broadcast their prefered update c_i to the other producers.

3.4.2 Candidate collection

P_i during this phase will be collecting the c_j values from other producers. At the end of this cycle P_i will hold a set V_i candidates.

3.5 Voting Phase

During the third phase ($a.k.a$ voting phase) of a ledger cycle, a producer P_j elects a partial ledger state update from the collection of producer candidates that it has received. At the end of the process, producers forward their vote which comprises a complete ledger state update including a reward to some producers.

The third phase starts at $t = t_v$ where $t_v = t_p + \Delta t_p + \Delta t_c$ and lasts for a period of time Δt_v, therefore ending at $t_v + \Delta t_v$.

3.5.1 Ballot generation and broadcast

At $t = t_v$:

1. P_j verifies that the same first hash value h^{maj} is embedded in a majority of producer candidates. With $h^{maj} = max[unique(h_{\Delta k}^{maj}) \; \forall \; k \in \{V_j\}]$ and $V^{maj} = count[(h_{\Delta k}^{maj} = h^{maj}) \; \forall \; k \in \{V_j\}]$, this condition is met if $V^{maj} > V_{threshold}$ (See section **??** for more explanations).
2. The producer P_j can only participate in the following steps if the local first hash value computed during the construction phase, $h_{\Delta j}$, is equal to h^{maj}. Indeed, P_j needs to have knowledge of the partial ledger state update of which the hash was used to vote in order to proceed.

If each producer collects the first hash value generated by every producer, any two producers P_j and P_k would build the same list of identifiers $\mathcal{L}_j(prod) = \mathcal{L}_k(prod)$. In practice, a producer may not have collected all P first hash values and as a result may have an incomplete list of identifiers, yet have collected enough data to be able to confidently issue a vote on the most popular partial ledger state update. We mentioned how the identifier of a producer can be appended to a first hash value to a) verify if P_j is a producer node selected for the ledger cycle and b) evaluate the quality of work performed by P_j. Indeed, Id_j can be used to create and add a compensation entry to the ledger state update, that rewards the producer for its work performed during the ledger cycle. The correct (complete) list of producers who successfully built the correct (most popular) partial ledger state update for that cycle, $\mathcal{L}_n(prod)$, is used to create these new transaction entries and append them to the final ledger state update generated for that cycle. It is therefore crucial that a majority of producers succeed in generating that list in order to generate the same complete ledger state update. A complete ledger state update should comprise the list of transaction entries and transaction signatures included in a partial ledger state update as well as the compensation entries rewarding the producers.

The voting process thus consists in creating the final list of identifiers involved in the production of the partial ledger state update. As explained below the final list $\mathcal{L}_n(prod)$ is obtained by merging the partial lists included in the producers' candidate. A producer P_j could have produced a first hash value $h_{\Delta j}$ different to $h^{maj}_{\Delta j}$ yet added his identifier to $\mathcal{L}_j(prod)$ when building its candidate c_j in the attempt to collect some token reward. In such scenario Id_j would be an element of the list included in c_j (or any other producer node controlled by P_j), but it wouldn't be included in any other list $\{\mathcal{L}_k(prod)\}\forall\, k \in P/j$. To prevent such malicious behaviour, a rule imposes that P_j only appends to the final list $\mathcal{L}_n(prod)$ the identifier of a producer included in the list $\mathcal{L}_k(prod)$ of a candidate c_k satisfying $h^{maj}_{\Delta k} = h^{maj}$ if and only if that identifier is included in at least $P/2$ lists $\{\mathcal{L}_k(prod)\}_{k=1,..,V_j}$ associated to a candidate c_k satisfying $h^{maj}_{\Delta k} = h^{maj}$. Only a producer controlling half or more of the producer nodes would succeed in including its identifier into the final list $\mathcal{L}_n(prod)$.

Although this eliminates the risk of unethical behaviour from the producer, this also means that there would be little incentive for a producer to broadcast its vote if its identifier was not included in $\mathcal{L}_n(prod)$. However, the probability that a producer compiles the correct final list $\mathcal{L}_n(prod)$ strongly depends on the number of votes collected. The more votes collected by a producer, the greater the probability that said producer will compile the complete final list. Although a producer may not have produced the correct partial ledger state update, participating in the voting process is, therefore, an important contribution to the overall consensus protocol and should entitle the producer nodes to some reward. To that end a producer P_j can use the identifier of other producers included in their vote and create a second list $\mathcal{L}_j(vote)$ to account for their participation in the voting process.

P_j follows a series of step for a period of time Δt_{v0} ($\Delta t_{v0} < \Delta t_v$):

1. P_j creates a new list $\mathcal{L}_j(vote)$ and appends to said list the identifier of any producer P_k who forwarded a candidate c_k satisfying $h^{maj}_{\Delta k} = h^{maj}$.
2. P_j creates the final list $\mathcal{L}_n(prod)$ and appends to said list the identifier of a producer included in the list $\mathcal{L}_k(prod)$ of a candidate c_k satisfying $h^{maj}_{\Delta k} = h^{maj}$ if and only if that identifier is included in at least $P/2$ lists $\{\mathcal{L}_k(prod)\}_{k=1,..,V_j}$ associated to a candidate c_k satisfying $h^{maj}_{\Delta k} = h^{maj}$.
3. P_j then creates a list L_{CE} of compensation entries for each producer whose identifier is included in $\mathcal{L}_n(prod)$. Each producer receives x_h tokens. Assume that $C_n \leq P$ identifiers are included in $\mathcal{L}_n(prod)$ and X is the total number of tokens injected per cycle for the pool of P producers. The quantity x_h is defined such that $C_n x_h = f_{prod}X + x_f$ where x_f represents the total number of fees collected from the m_{n-1} transactions and f_{prod} represents the fraction of new tokens injected per cycle and distributed to the producers who built the correct ledger state update. The remaining $(1 - f_{prod})X$ tokens are distributed to other contributing nodes in the network. A part of this remainder goes to the producers who voted correctly on the previous ledger cycle update. Let $\mathcal{L}_{n-1}(vote)$ be the list of the identifiers of producers who voted correctly on the previous ledger cycle update $\mathcal{C}_{n-1}$. We later demonstrate how such a list is derived during a ledger cycle. For now, let's assume that L_{CE} includes compensation entries for producers involved the production of the ledger state update for this ledger cycle $\mathcal{C}_n$ and the producers involved in the voting process of the preceding cycle $\mathcal{C}_{n-1}$.
4. P_j then creates the candidate ledger state update for $\mathcal{C}_n$ including the reward allocated to the producers for their contribution:

$$\mathbf{LSU_j = L^f_E \,||\, d_n \,||\, L_{CE}}$$

   P_j then computes its vote (or *producer vote*):

$$\boxed{v_j = \mathcal{H}(LSU_j) \,||\, \#(\mathcal{L}_j(vote)) \,||\, Id_j} \tag{1}$$

   which includes the hash of the candidate ledger state update (or *second hash value*) and a partial list of identifiers of producers who designated the correct candidate partial ledger state update corresponding to h^{maj}.
5. P_j then forwards v_j to the other producers and collects the producer votes issued by its peers. Figure **??** illustrates the different steps followed by P_j during the voting phase.

3.5.2 Ballot collection

During the voting phase, the producer P_j collects the producer votes broadcast by its peers. At the end of the voting phase ($t = t_v + \Delta t_v$), the producer P_j holds U_j producer votes in its cache with $U_j \leq C_n$ where $C_n \leq P$ is the actual total number of producers who correctly computed h^{maj}.

3.6 Synchronisation Phase

References

[1] J. Benet, "Ipfs-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.