Distributed Registry Protocol - DRiP

Harsha Bellur
Chris Wendt
Overview

• DRiP is a HTTP based protocol for sharing registry type of information between interconnected nodes across a network

• It uses a gossip protocol for complete distribution across interconnected nodes

• It incorporates a voting mechanism to avoid conflicting data updates or race conditions
Overview

• It is designed to use full reliability in its transactions, even at the price of inefficiency. Speed or optimization of transactions is only secondary consideration.

• Conflicting updates, will need to be fully retried.

• Synchronization transactions are comprehensive.

• Conflicts, assuming each participant is managing mostly its own data, other than change of ownership and other relatively rare occurrences, should happen very infrequently

• Regardless, assuming ~100-1000 node distributed mesh, updates across nodes should be updated in ~seconds time frame assuming all nodes are well connected
Distributed Mesh
Distributed Mesh

• A distributed mesh is a network topology of interconnected nodes that share data.

• There is no assumption of a fully connected mesh, but in general most nodes SHOULD have at least 2 connections to other nodes for redundancy and gossip propagation purposes.

• Gossip protocol and the use of a counter allows for all nodes in the mesh to receive updates from all other nodes.
Transactions

- Two basic transactions
  - Update - A node has new or modified key-value data and would like to update peer nodes
  - Sync - A node is either newly established or was in an inactive state for a period of time and requests a peer to provide a full update of data to make sure it is fully synchronized with network.
Node State

- Each node should maintain a state of:
  - active
  - inactive
  - sync
Node State - API

POST /node/:nodeid/active

POST /node/:nodeid/inactive

GET /state
Node-specific info

- Each node should have a globally unique ID
- Node information and other transaction specific info is carried in custom HTTP header fields
Custom HTTP Header Fields

- DRiP-Node-ID
- DRiP-Node-Counter
- DRiP-Node-Counter-reset
- DRiP-Transaction-Type
- DRiP-Sync-Complete
Key-Value Data Propagation Rules

• For update transactions, nodes should propagate key-value data to it’s peer nodes except the node that it received data

• For all transactions,
  • DRiP-Node-ID and associated latest DRiP-Node-Counter values should be recorded by receiver nodes
  • If DRiP-Node-Counter is greater for a particular DRiP-Node-ID the key-value data should be recorded and propagated to peer nodes.
  • DRiP-Node-Counter-reset provide a wrap-around mechanism for DRiP-Node-Counter as a forever incrementing integer
Voting and Commit Phases for Update

• When initiator node has new data, it initiates an Update

• Update consists of a two-phase commit procedure to avoid race conditions or potential error conditions

• Two phases are called:
  • voting phase
  • commit phase
Voting and Commit Phases for Update

1. **Waiting For Events**
   - (Update, Start Timer)

2. **Waiting For Response From Peer Nodes**
   - Timer Expired
   - Received Update From Peer Node
   - Received Votes From All Peer Nodes

3. **Validating Votes**
   - If key matches an in-progress update vote "no".
   - Otherwise, vote "yes".

4. (If all Votes are "YES", propagate commit)
Voting Phase

- An initiator node first sends a voting request to all of its peer nodes.
- Only the initiator node sets a timer for the voting phase to time out if it hasn’t received responses from all its peer nodes.
- As in gossip, each receiver node will continue propagating voting requests to its peer nodes until it has received a voting query again.
- If all voting requests result in “yes”, the commit phase can be initiated by initiator node.
Voting Phase - API

POST /voting

POST /votingphase/node/:nodeid/response/:response
Commit Phase

• An initiator node, upon successful voting phase, commits the key-value data to its local data store and sends a commit request to its peers.

• Each receiver node commits data to its local data store and, again as in gossip, propagates the commit request to each of its peer nodes, until it receives a duplicate commit request and stops.
Commit Phase - API

POST /commit
Node Sync

• When a new node is added to a distributed registry, or when a node has been offline or inactive for any period of time, a node sync operation must be completed.

• The node should go to active state to receive updates concurrently with sync operations.

• The node will make a sync request to one of its peer nodes

• The peer node will then sync a complete set of key-value data in the form of commit requests (no voting phase)
Node Sync

- Best practice in terms of rate limiting, priority scheduling, and others should be employed to avoid overwhelming connections or risk interfering with update transactions.

- Best practice for redundancy and fail-over should be followed to avoid as much occurrence of inactive time that requires sync operations.
Sync - API

PUT /sync/node/:nodeid
Heartbeat

- TBD in next update

- Main issue is best methods to determine how to proceed if a peer node doesn’t send heartbeat and didn’t declare itself “inactive”
Authentication/Entitlement

- Took the approach that scope of this spec only has protocol for exchanging data
- Assumes any authentication or entitlement of write/read capability or permissions sits a layer above this protocol and/or in the key-value data model