aptos No.8 consensus_proposal

https://github.com/xiaoyu1998/

Resolution:2560×1440(2K)
https://github.com/aptos-labs/aptos-core/tree/aptos-node-v1.28.7

Entries

main
aptos-node/src/main.rs

```
AptosNodeArgs::parse().run()
```

run
aptos-node/src/lib.rs

```
start(config, None, true).expect("Node should start correctly");
```

start_and_report_ports
aptos-node/src/lib.rs

```
let _node_handle = setup_environment_and_start_node(
```

setup_environment_and_start_node
aptos-node/src/lib.rs

```
let mut admin_service = services::start_admin_service(&node_config);
storage::initialize_database_and_checkpoints(&mut node_config)?;
admin_service.set_aptos_db(db_rw.clone().into());
utils::set_aptos_vm_configurations(&node_config);
let telemetry_runtime = services::start_telemetry_service(
create_event_subscription_service(&node_config, &db_rw);
network::create_peers_and_metadata(&node_config);
network::setup_networks_and_get_interfaces(
services::start_peer_monitoring_service(
state_sync::start_state_sync_and_get_notification_handles(
services::start_node_inspection_service(
services::bootstrap_api_and_indexer(
admin_service.set_mempool_client_sender(mempool_client_sender);
services::start_mempool_runtime_and_get_consensus_sender
consensus::create_dkg_runtime
consensus::create_jwk_consensus_runtime
state_sync_runtimes.block_until_initialized();
consensus::create_consensus_observer_and_publisher(
consensus::create_consensus_runtime(
```

epoch_manager

create_consensus_runtime
aptos-node/src/consensus.rs

```
let (consensus_runtime, consensus_db, quorum_store_db) =
services::start_consensus_runtime(
admin_service.set_consensus_dbs(consensus_db, quorum_store_db);
consensus_runtime
```

start_consensus_runtime
aptos-node/src/services.rs

```
let consensus =
aptos_consensus::consensus_provider::start_consensus(
```

start_consensus
consensus/src/consensus_providers.rs

```
let execution_proxy = ExecutionProxy::new(
let execution_client = Arc::new(ExecutionProxyClient::new(
let epoch_mgr = EpochManager::new(
NetworkTask::new(network_service_events, self_receiver);
runtime.spawn(network_task.start());
runtime.spawn(epoch_mgr.start(timeout_receiver, network_receiver));
```

start
consensus/src/epoch_manager.rs

```
self.await_reconfig_notification().await;
loop {
    network_receivers.consensus_messages.select_next_some() => {
        self.process_message(peer, msg).await
    network_receivers.quorum_store_messages.select_next_some()
        self.process_message(peer, msg).await
    network_receivers.rpc_rx.select_next_some()
        self.process_message(peer, msg).await
    round = round_timeout_sender_rx.select_next_some()
        self.process_local_timeout(round)}
```

await_reconfig_notification
consensus/src/epoch_manager.rs

```
self.start_new_epoch(reconfig_notification.on_chain_configs)
```

start_new_epoch
consensus/src/epoch_manager.rs

```
self.start_new_epoch_with_jolteon(
```

round_manager

start_new_epoch_with_jolteon
consensus/src/epoch_manager.rs

```
match self.storage.start(
    self.start_round_manager(
```

start_round_manager
consensus/src/epoch_manager.rs

```
self.execution_client.start_epoch(
let proposal_generator = ProposalGenerator::new(
let mut round_manager = RoundManager::new(
tokio::spawn(round_manager.start(round_manager_rx,
buffered_proposal_rx, close_rx));
```

start
consensus/src/round_manager.rs

```
loop {
    proposal = buffered_proposal_rx.select_next_some() => {
        for proposal in proposals {
            self.process_proposal_msg(*proposal_msg).await
    (peer_id, event) = event_rx.select_next_some() => {
        VerifiedEvent::VoteMsg(vote_msg) => {
            self.process_vote_msg(*vote_msg)
        VerifiedEvent::RoundTimeoutMsg(timeout_msg) => {
            self.process_round_timeout_msg(*timeout_msg)
        VerifiedEvent::OrderVoteMsg(order_vote_msg) => {
            self.process_order_vote_msg(*order_vote_msg)
```

process_vote_msg
consensus/src/round_manager.rs

```
self.process_vote(vote_msg.vote())
```

process_vote
consensus/src/round_manager.rs

```
self.process_vote_reception_result
```

process_vote_reception_result
consensus/src/round_manager.rs

```
VoteReceptionResult::NewQuorumCertificate(qc) => {
    self.new_qc_aggregated(qc.clone(), vote.author())
        self.broadcast_order_vote(vote, qc.clone())
VoteReceptionResult::New2ChainTimeoutCertificate(tc) => {
    self.new_2chain_tc_aggregated(tc)
VoteReceptionResult::EchoTimeout(_)
    self.process_local_timeout(round).await
```

proposal_generator

new_qc_aggregated
consensus/src/round_manager.rs

```
.block_store.insert_quorum_cert(&qc, &mut
self.create_block_retriever(preferred_peer))
self.process_certificates().await?;
```

process_certificates
consensus/src/round_manager.rs

```
.round_state.process_certificates(sync_info, &epoch_state.verifier)
.process_new_round_event(new_round_event).await?;
```

process_new_round_event
consensus/src/round_manager.rs

```
if is_current_proposer {
    Self::generate_and_send_proposal(
```

generate_and_send_proposal
consensus/src/round_manager.rs

```
let proposal_msg = Self::generate_proposal(
network.broadcast_proposal(proposal_msg).await;
```

generate_proposal
consensus/src/round_manager.rs

```
proposal_generator
.generate_proposal(new_round_event.round, proposer_election, callback)
Block::new_proposal_from_block_data_and_signature(proposal, signature);
observe_block(signed_proposal.timestamp_usecs(), BlockStage::SIGNED);
```

generate_proposal
consensus/src/liveness/proposal_generator.rs

```
let (validator_txns, mut payload) = self
    .payload_client.pull_payload(
let block = if self.vtxn_config.enabled() {
    BlockData::new_proposal_ext(
} else {
    BlockData::new_proposal(
```

pull_payload
consensus/src/payload_client/mixed.rs

```
self.user_payload_client
    .pull(user_txn_pull_params, wait_callback)
```

epoch_manager => round_manager => proposal_generator => playload_client => VTxnPoolState