FOROS
Failover ROS Framework

Wonguk Jeong (wonguk.jeong@42dot.ai)
Overview
Motivation

Reliability and Safety are essential for autonomous vehicles and robots.
How?

Mitigating Single Points of Failure (SPOFs)

- Control
- Planner
- Perception
- Sensor
- System Health Monitor

Failed

SPoF
So, How?

Enhancing high availability of safety-critical modules using **redundancy** (= clustering)

**Control**

**Planner**

**Perception**

**Sensor**

**System Health Monitor**

<table>
<thead>
<tr>
<th>State</th>
<th>System Health Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active</td>
<td></td>
</tr>
<tr>
<td>Standby</td>
<td>System Health Monitor</td>
</tr>
<tr>
<td>Standby</td>
<td>System Health Monitor</td>
</tr>
</tbody>
</table>
Failover ROS Framework

An open source ROS2 framework that can be used to provide redundancy for safety-critical nodes using a RAFT consensus algorithm with minimal effort.
Constraint

This framework can tolerate failures equal to the cluster size minus quorum.

<table>
<thead>
<tr>
<th>Cluster Size (N)</th>
<th>Quorum (Q = N / 2 + 1)</th>
<th>Number of fault tolerant nodes (N - Q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

This framework tolerates **fail-stop failure** but NOT **Byzantine failure**

- **Fail-stop failure**: the component stops operating.
- **Byzantine failure**: there is imperfect information on whether a component has failed
Core Features

- **Leader election**: determination of active nodes by election
- **Log replication**: consensus-based data storage. Mainly used for state replication
- **Inspector**: a tool for monitoring the status of clusters.

```
Log replication = State replication
```

```
A¹
Active

A²
Standby

A³
Standby
```

```
Failed

A¹
Active

A²
Standby

A³
Standby
```

```
State

State

State
```

```
State

State

State
```

```
State restoration
```
Leader Election

All nodes have one of the following states: 'Follower', 'Candidate', or 'Leader'

- Election timeout, starts election
- Receives votes from majority
- Discovers new leader
Leader Election

All nodes start in ‘Follower’ state

A¹
<br/><span style="color: #2b8b34;"><Follower></span>

A²
<br/><span style="color: #2b8b34;"><Follower></span>

A³
<br/><span style="color: #2b8b34;"><Follower></span>
Leader Election

If a ‘Follower’ does not receive a ‘Leader’ heartbeat for a certain period of time, it is changed to ‘Candidate’ and an election is held.

1. Election timeout, starts election

   ![Diagram](https://example.com/diagram.png)

   - **A¹**: <Candidate>
   - **A²**: <Follower>
   - **A³**: <Follower>

   2. Vote for me
   3. Request a vote
Leader Election

When a ‘Candidate’ receives a majority of the votes, it becomes the ‘Leader’.

1. Get a vote
2. Received votes from majority
Leader Election

The 'Leader' periodically sends heartbeats to prevent elections for new leaders.

1. Send heartbeat
Leader Election

The complex leader election process is all handled within the FOROS framework. Developers only need to consider ‘Standby’ and ‘Active’ states.

Transitioned to ‘Leader’ state

Transitioned to ‘Follower’ state

Filter the messages below:
- Published topics
- Received service requests
Leader Election: How to Use

Simple! Use `ClusterNode` class instead of `rclcpp::Node`.

```cpp
auto node = akit::failover::foros::ClusterNode::make_shared(
    "Test_cluster", // Cluster Name
    0,              // Node ID
    std::initializer_list<uint32_t>{0, 1, 2} // Node IDs in the given cluster
);

node->register_on_activated([&]() { RCLCPP_INFO(logger, "activated"); });
node->register_on_standby([&]() { RCLCPP_INFO(logger, "standby"); });
```

Register state transition callbacks using `register_on_activated`, `register_on_standby`
Log Replication

When the ‘Leader’ requests to store data, it requests data synchronization from other nodes and succeeds when more than half of the nodes are synchronized.
Log Replication : How to Use

Use `commit_command` to request to store data

```
node->commit_command(
    akit::failover::foros::Command::make_shared(std::initializer_list<uint8_t>{
        1}),
    [&](akit::failover::foros::CommandCommitResponseSharedFuture
        response_future) {
        auto response = response_future.get();
        if (response->result() == true) {
            RCLCPP_INFO(logger, "commit completed");
        } else {
            RCLCPP_ERROR(logger, "commit failed");
        }
    });
```
Log Replication : How to Use

Use `get_commands_size`, `get_command` to get stored data.

```cpp
int len = node->get_commands_size();
auto command = get_command(len - 1);
```

Use `register_on_committed`, `register_on_reverted` to register commit/revert callback.

```cpp
node->register_on_committed(
    [&](int64_t id, akit::failover::foros::Command::SharedPtr command) {
        RCLCPP_INFO(logger, "command commited : %ld, %d", id, command->data()[0]);
    });

node->register_on_reverted([&](int64_t id) {
    RCLCPP_INFO(logger, "command reverted until : %ld", id);
});
```
Inspector

Visualize active cluster information and node information in the cluster with TUI

<table>
<thead>
<tr>
<th>Summary</th>
<th>Cluster Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Cluster Name</td>
</tr>
<tr>
<td>Size</td>
<td>Cluster Size</td>
</tr>
<tr>
<td>Term</td>
<td>Election No.</td>
</tr>
<tr>
<td>Active</td>
<td>Active Node IDs</td>
</tr>
<tr>
<td>Leader</td>
<td>Leader ID</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Details</th>
<th>Node Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node ID</td>
<td>Node ID</td>
</tr>
<tr>
<td>State</td>
<td>State</td>
</tr>
<tr>
<td>Term</td>
<td>Election No.</td>
</tr>
<tr>
<td>Voted For</td>
<td>ID</td>
</tr>
<tr>
<td>Data Size</td>
<td>Stored Data Size</td>
</tr>
<tr>
<td>Size</td>
<td>Cluster Size</td>
</tr>
</tbody>
</table>
Links

- FOROS Github: https://github.com/42dot/foros
- FOROS Wiki: https://github.com/42dot/foros/wiki
- RAFT: https://raft.github.io/
- 42dot: https://42dot.ai/