

ReCraft Artifact

ReCraft Artifact consists of two parts:

I. Executable ReCraft Go Code.

II. Formal specifications and proof sketches of ReCraft protocols in Coq/Rocq.

With I, one can test, run, and reuse ReCraft executable code, and with II, one may learn how ReCraft is formally specified and what are the key invariants and proof structures of the protocol. For II, successfully compiling the project means the interactive theorem prover accepts what is specified and proved. This document repeats the README files of each part in separate directories.

I. ReCraft Go Code

ReCraft is implemented based on etcd. ReCraft adds cluster split and merge functionality as well as single cluster membership change scheme that improves vanilla etcd. The code is a fork of etcd and everything in etcd is also included in the package. This document describes how to install and test ReCraft. The names of example scripts are given after describing the basic operations. The scripts are provided as separate files and they include more realistic examples with multi-node settings. Although this document uses examples that uses multiple instances of etcd on a single node, by deploying etcd instances on distributed nodes and changing the IP/Port pairs to corresponding deployment would enable running ReCraft in a truly distributed setting.

Besides the basic workings of ReCraft in this document, the evaluation directory in the package contains scripts that we used to deploy and evaluate ReCraft on our distributed cluster. These scripts are used to collect numbers that are used to generate the graphs in the paper.

1. ReCraft Cluster Setup Guide

The installation process for ReCraft is pretty much the same as installing etcd.

Pre-requisites

Install Go

Follow the instructions to install Go: <https://go.dev/doc/install>.

Install Python

Ensure Python 3.13 or higher is installed. You can check your Python version with:

```
python3 --version
```

If Python is not installed, download it from python.org or use your system's package manager.

Set Up a Virtual Environment

Create a virtual environment to isolate dependencies:

```
python3 -m venv .venv
```

Activate the virtual environment:

- On macOS/Linux:

```
source .venv/bin/activate
```

- On Windows:

```
.venv\Scripts\activate
```

Install Required Python Packages

Install `fabric` and `requests` using `pip`:

```
pip3 install fabric requests
```

Verify the installation:

```
pip3 show fabric requests
```

Setting Up Your Environment

Install `gobin`

```
go install github.com/myitcv/gobin@latest
```

Add Go Binaries to PATH

```
export PATH=$PATH:$(go env GOPATH)/bin
```

Building the Binaries

Build `etcd` and `etcdctl`

At the root of the repository, run:

```
make build
```

Set `LOCAL_ETCD_DIR` Environment Variable

Set the `LOCAL_ETCD_DIR` environment variable to the root of the repository:

```
export LOCAL_ETCD_DIR=$(pwd)
```

This ensures that scripts and tools referencing this variable can locate the repository's root directory.

Add `etcd` and `etcdctl` to PATH

At the root of the repository, run:

```
export PATH=$PATH:$LOCAL_ETC_DIR/bin
```

Build the Server Binary

Navigate to the `server` directory and run:

```
go build -o server
```

2. Starting a Cluster

The following is a basic methodology to deploy a small cluster.

Start a 3-Node Cluster

Navigate to the `deploy` directory (which contains the `fabfile`) and run:

```
fab start 1=http://127.0.0.1:1380,2=http://127.0.0.1:2380,3=http://127.0.0.1:3380
```

3. Managing Clusters using Split and Merge features of ReCraft

Split a Cluster into Multiple Subclusters

First, get member IDs(HEX IDs) using:

```
etcdctl --write-out=table member list
```

Then split the cluster:

```
etcdctl --endpoints=<endpoints> member split <member_ids_group1> <member_ids_group2>
```

Example:

```
etcdctl --  
endpoints=http://127.0.0.1:1380,http://127.0.0.1:2380,http://127.0.0.1:3380,http://127.0.0.  
.1:4380,http://127.0.0.1:5380 member split  
7ac641502b72a71a,b71f75320dc06a6c,d07d5325fff892c1 b7bacd4212cc9323,a100ada638d79265
```

Merge Multiple Subclusters into a Single Cluster

```
etcdctl member merge <cluster_member_url1>,<cluster_member_url2>
```

Example:

```
etcdctl member merge http://127.0.0.1:2380,http://127.0.0.1:4380
```

Testing Split and Merge Functionality

You can test the split and merge functionality using the provided script at the root:

```
./test_sm.sh
```

4. Single Cluster Membership Change (i.e., Adding and Removing Nodes)

The ReCraft code package includes three different reconfiguration schemes in the ReCraft paper (DSN 2025): (1) ReCraft single cluster membership change, (2) Raft one-at-a-time single cluster membership change (AR-RPC), and (3) Raft joint consensus single cluster membership change (JC). However, (1) subsumes (2) so we introduce how to run (1) and (3). To trigger (2) one only needs to add/remove one node at a time using approach (1).

(1) Using ReCraft Single Cluster Membership Change (subsumes (2))

To use ReCraft single cluster membership change, one may go through one or two steps. When adding or removing one node, simply adding and removing suffices and this is the same as Raft one-at-a-time (AR-RPC) approach. When adding two nodes to even numbered cluster, simply adding the nodes suffices. When adding/removing two or more nodes in other cases, one must first enter joint mode by adding/removing nodes and then leave joint mode. As the paper describes ReCraft can only remove nodes fewer than the quorum size of the old configuration.

Add a Node:

```
etcdctl member joint --add <node_url> --mode recraft
```

Example:

```
etcdctl member joint --add http://127.0.0.1:4380 --mode recraft
```

Script: You can test the functionality of adding a single node using the ReCraft consensus with the provided script located at the root:

```
./test_scm_add_single.sh
```

Add Multiple Nodes:

```
etcdctl member joint --add <node_url1>,<node_url2> --mode recraft
```

Example:

```
etcdctl member joint --add http://127.0.0.1:4380,http://127.0.0.1:5380 --mode recraft
```

Leave Joint Consensus:

```
etcdctl member leave joint
```

Script: You can test the functionality of adding multiple nodes using the ReCraft consensus with an explicit leave joint command by using the provided script located at the root:

```
./test_scm_add_multiple_explicit_leave.sh
```

You can test the functionality of adding multiple nodes using the ReCraft consensus with an implicit leave joint command by using the provided script located at the root:

```
./test_scm_add_multiple_implicit_leave.sh
```

Remove a Node:

First, get member IDs(HEX IDs) using:

```
etcdctl --write-out=table member list
```

```
etcdctl --endpoints=<endpoints> member joint --remove <member_id> --mode recraft
```

Example:

```
etcdctl --endpoints=http://127.0.0.1:1380,http://127.0.0.1:2380,http://127.0.0.1:3380  
member joint --remove d07d5325fff892c1 --mode recraft
```

Script: You can test the functionality of removing a single node using the ReCraft consensus with the provided script located at the root:

```
./test_scm_remove_single.sh
```

Remove Multiple Nodes: First, get member IDs(HEX IDs) using:

```
etcdctl --write-out=table member list
```

```
etcdctl --endpoints=<endpoints> member joint --remove <member_id1>,<member_id2> --mode  
recraft
```

Example:

```
etcdctl --  
endpoints=http://127.0.0.1:1380,http://127.0.0.1:2380,http://127.0.0.1:3380,http://127.0.0.  
.1:4380,http://127.0.0.1:5380 member joint --remove b7bacd4212cc9323,a100ada638d79265 --  
mode recraft
```

Leave Joint Consensus:

```
etcdctl member leave joint
```

Script: You can test the functionality of removing multiple nodes using the ReCraft consensus with an explicit leave joint command by using the provided script located at the root:

```
./test_scm_remove_multiple_explicit_leave1.sh
```

```
./test_scm_remove_multiple_explicit_leave2.sh
```

(3) Using Raft Joint Consensus Single Cluster Membership Change

To use Raft joint consensus single cluster membership change, one should always go through two steps: first enter joint mode by adding/removing nodes and then leave joint mode.

Add a Node:

```
etcdctl member joint --add <node_url>
```

Example:

```
etcdctl member joint --add http://127.0.0.1:4380
```

Script: You can test the functionality of adding a single node using the Raft joint consensus with the provided script located at the root:

```
./test_joint_add_single.sh
```

Add Multiple Nodes:

```
etcdctl member joint --add <node_url1>,<node_url2>
```

Example:

```
etcdctl member joint --add http://127.0.0.1:4380,http://127.0.0.1:5380
```

Script: You can test the functionality of adding multiple nodes using the Raft joint consensus by using the provided script located at the root:

```
./test_joint_add_multiple.sh
```

Remove a Node:

First, get member IDs(HEX IDs) using:

```
etcdctl --write-out=table member list
```

```
etcdctl --endpoints=<endpoints> member joint --remove <member_id>
```

Example:

```
etcdctl --endpoints=http://127.0.0.1:1380,http://127.0.0.1:2380,http://127.0.0.1:3380  
member joint --remove d07d5325fff892c1
```

Script: You can test the functionality of removing a single node using the Raft joint consensus with the provided script located at the root:

```
./test_joint_remove_single.sh
```

Remove Multiple Nodes: First, get member IDs(HEX IDs) using:

```
etcdctl --write-out=table member list
```

```
etcdctl --endpoints=<endpoints> member joint --remove <member_id1>,<member_id2>
```

Example:

```
etcdctl --  
endpoints=http://127.0.0.1:1380,http://127.0.0.1:2380,http://127.0.0.1:3380,http://127.0.0.  
.1:4380,http://127.0.0.1:5380 member joint --remove b7bacd4212cc9323,a100ada638d79265
```

Script: You can test the functionality of removing multiple nodes using the Raft joint consensus by using the provided script located at the root:

```
./test_joint_remove_multiple.sh
```

Leave Joint Consensus:

```
etcdctl member leave joint
```

This guide covers the essential commands for setting up, splitting, merging, and managing nodes in an ETCD cluster using both **ReCraft** and **Raft** consensus models. Customize the endpoints and member IDs to match your environment.

II. ReCraft Proofs

This artifact includes a prototype implementation of the ReCraft protocol along with proof sketches developed in Coq. Its primary goal is to reduce human error when demonstrating the safety properties of our intrinsic distributed consensus protocol—a part where mistakes often occur despite careful checks. In this sense, this artifact does not provide a complete end-to-end formal proof. Instead of that, it offers mechanized prototypes and several key invariants that play a central role in the safety arguments. These components are intended to verify whether the critical properties are correctly specified and well-formed.

How to Compile the Artifact

We highly recommend using the [opam package manager](#) to replicate our artifact. Once you have installed opam on your machine, follow the instructions below:

1. Update opam

```
opam update
opam upgrade
```

2. Install Coq

```
opam switch create reconfiguration ocaml-base-compiler 4.14.1
opam pin add coq 8.16.1
```

3. Add Required Repositories

```
opam repo add coq-released https://coq.inria.fr/opam/released
opam repo add iris-dev https://gitlab.mpi-sws.org/iris/opam.git
```

4. Install Required Packages

```
opam pin add coq-stdpp 1.8.0
opam install coq-metacoq-template.1.1+8.16
```

This will install all dependencies required for our artifact.

5. Compile the Project

Once all dependencies are installed, compile the project using:

```
make
```

File structures

Structure

The entire structure is as follows


```
src -- Prelude.v
    | - Clock.v
    | - RaftInterface.v
    | - InvariantDef.v
    | - Lens.v
    | - LocalToGlobal.v
    | - Network.v
    | - PreludeSimpl.v
    | - TC.v
    | - Raft.v
```

Key files

Network.v

It includes network definitions, which also reflect our assumptions about the network—namely, that it is benign but asynchronous.

Raft.v

Prototype sketch of ReCraft using Coq.

RaftInterface.v

We specified the pre- and post-conditions of the ReCraft interfaces by referencing the Raft.v file.

InvariantDef.v

This file contains definitions of several invariants.

LocalToGlobal.v

It includes several definitions used to explore how local invariants can be connected to global invariants—an essential aspect of proving ReCraft's safety.