Canister Profiling

Date	
Slides	
📓 Courses	IDP
Lecture video link	

trying out profiling the canisters using:

```
.
├── Cargo.lock
├── Cargo.toml
├── collections
│   ├── Makefile
│   ├── motoko
│   ├── perf.sh
│   ├── README.md
│   └── rust
├── dapps
│   ├── basic_dao.sh
│   ├── Makefile
│   ├── motoko
│   ├── nft.sh
│   ├── README.md
│   └── rust
├── heartbeat
│   ├── Makefile
│   ├── motoko
│   ├── perf.sh //This file is used to profile the canister function calls
│   ├── README.md
│   └── rust
├── Makefile
├── motoko
│   ├── classes.sh
│   ├── dfx.json
│   ├── gc.sh
│   ├── Makefile
│   ├── README.md
│   └── src
├── _out
│   ├── collections
│   ├── dapps
│   └── heartbeat
├── prelude.sh
├── pub-sub
│   ├── Makefile
│   ├── motoko
│   ├── README.md
│   └── rust
├── README.md
```

How this works is that we add a new benchmark folder where we add the following folder structure:

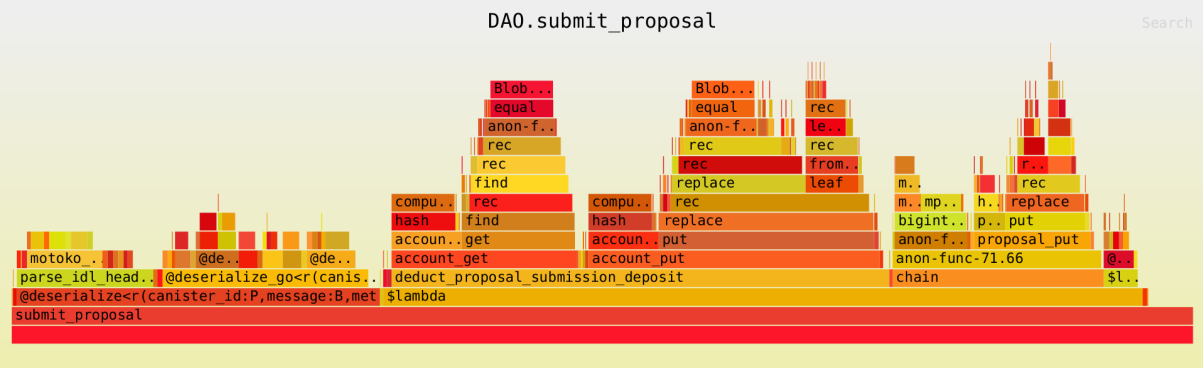
```
Benchmark_name/
  Makefile
  README.md // Perf result will be appended to this markdown file.
  perf.sh   // ic-repl script that generates perf result. If the candid interface is different, we can use multiple scripts.
  motoko/
    dfx.json
    src/
      benchmark1.mo
      benchmark2.mo
  rust/
    dfx.json
    benchmark1/
      Cargo.toml
```

```
    benchmark1.did
    src/
      lib.rs
  benchmark2/
    Cargo.toml
    benchmark2.did
    src/
      lib.rs
```

Steps:

1. There are two folders called `rust` and `motoko`. So we write down the canister code in them
2. dfx.json file defines all the canister runtime code location
3. The `perf.sh` does the profiling for the canister code.

Output:



We can profile each and every function calls that are taking place in the canister.

Issues while installing

1. It requires to specifically install `ic-repl` which is a programming language for ic ecosystem.
   - In order to install it, you need to download the binaries from their github https://github.com/dfinity/ic-repl.
   - it is downloaded as `ic-repl-linux-x86` executable file.
   - Rename the file to `ic-repl` make the file executable and add it to the path
2. once you have installed it, it requires `dfx` to be running on port 4939. (not sure why)
   - by default, dfx runs at 8080
   - To change it, we have to reconfigure `dfx` network.
3. It also requires rust to be downloaded and installed.
   - basic installation won't work as we require wasm32 support for rust.
   - `rustup target add x86_64-unknown-linux-gnu` and `rustup target add wasm32-unknown-unknown` must be enabled in order to start the wasm32 compilation.