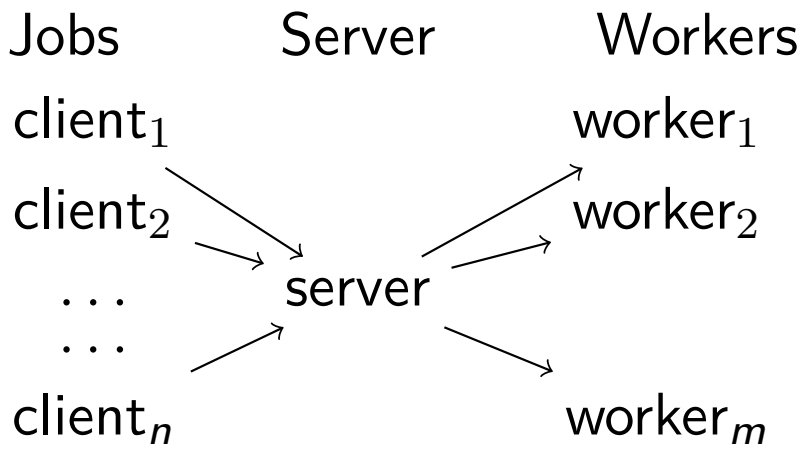
jobqueue: job stealing everywhere

https://github.com/thierry-martinez/jobqueue

Thierry Martinez (SED)

Developer Meetup, 30 May 2023

Job stealing



Job stealing is central in CI

- GitLab-CI acts as a job stealing server, where jobs come from CI pipelines and workers are gitlab-runner instances.
- GitHub actions acts as a job stealing server, where jobs come from CI workflows and workers are GitHub-hosted or self-hosted runners.
- What if jobs or self-hosted runners are spawned dynamically? What if self-hosted runners are private resources?

The jobqueue tool

Run a server on a VM:

```
$ jobqueue-server localhost:3821
```

Run workers:

```
$ jobqueue-worker server:3821
```

Run jobs:

```
$ jq command-line
```

What do the jq tool does?

The `jq` tool:

- executes `command-line` on the first ready worker,
- redirects standard input, standard output and standard error, signals (jobs can be interrupted by `Ctrl-C`),
- waits that the job is completely executed and terminates with the same exit code than the job itself,
- if the worker disconnects before the job is terminated, the job is requeued completely.

In other words, `jq` mimicks the execution of `command-line`, but the execution is deported on a worker.

Use cases

- `jobqueue-server` on a GitHub self-hosted runner, a first job in CI workflow runs `jobqueue-worker` with `slurm` on Cleps cluster, on other jobs run `jq`;
- `jobqueue-worker` can be run on `ci.inria.fr` CloudStack with dynamically allocated VMs;
- `jobqueue-worker` can be run N times locally and `jq` acts as a semaphore with parameter N.

Implementation

- Implemented in Rust, convenient and natural for such a combination of system programming and concurrency;
- Uses ∅MQ sockets, using serde for translating data-structures into network messages;
- Uses traits for mocking clients, servers and workers.

Protocol described in datatypes

```
pub trait Server {
    type Job: Job;
    type Run: Run;
    fn queue(&mut self, command_line: CommandLine)
      -> Self::Job;
    fn query(&mut self,
      callback: Box<dyn FnOnce(Query<Self::Run>)>);
}

#[derive(Clone, serde::Serialize, serde::Deserialize)]
pub enum ServerCommand {
    Queue(CommandLine),
    Query
}
```

Thank you!