FC

Release 0.6.11

Larry Shen

Dec 25, 2023

CONTENTS

CHAPTER

ONE

WHAT IS FC?

FC (Framework Coordinator), is an open source coordinator service among different embeded frameworks. It's built using the same principles as Mesos, provides different frameworks (e.g., LAVA, Labgrid) with API's for resource management and scheduling across entire board farms.

CHAPTER

TWO

BACKGROUND

There are many different systems in embedded test community, see this. They are good at different scenarios, e.g.

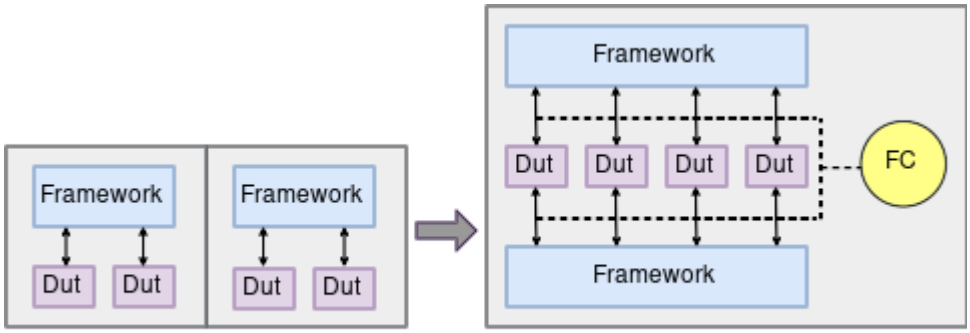
- **Test automation:**

  LAVA, Fuego, autotest, avocado, TI VATF, U-Boot test tool, CI-RT R4D, Baylibe Lab in a Box……

- **Development automation:**

  Labgrid……

Let's take a look at why FC is useful by going back in time.



- **Traditional deployment era:** Early on, as nearly all above systems require dedicated control of the board resources, organizations have to afford multiple series of hardware to meet the requirements of different frameworks, which leads to resource waste when leverage different systems.
- **FC deployment era:** As a solution, FC was introduced. It allows different frameworks could share same series of hardware, move resources between systems seamlessly. Different systems will continue work without aware existence of other systems. So, FC could be treat as "scheduler's scheduler". The initial idea comes from Jan's proposal to move boards between LAVA and labgrid on ATS2019, we realized the FC based on that thought (Thanks community to inspire the idea).

Compared to other resource management system, it has low invasion to frameworks. This means: unlike mesos which you should write your mesos scheduler for your framework & inject it into your framework code, FC won't change your framework code, it tries to adapt to your framework. But still, your framework need next two features:

- Your framework needs to have a job queue which FC could monitor.
- Your framework needs to have ways to let FC control resource's availability, temporary connect/disconnect from your framework.

CHAPTER

THREE

SUPPORTED FRAMEWORKS

FC is designed as a plugin system to support different frameworks, it currently supports:

- **LAVA:** use it for test automation
- **Labgrid:** use it for development automation

But, it not limits to above two, you could write your own plugins to support other framework.

Check out the *Overview* section for further information, also go to *Installation* for how to install this project.

CHAPTER

FOUR

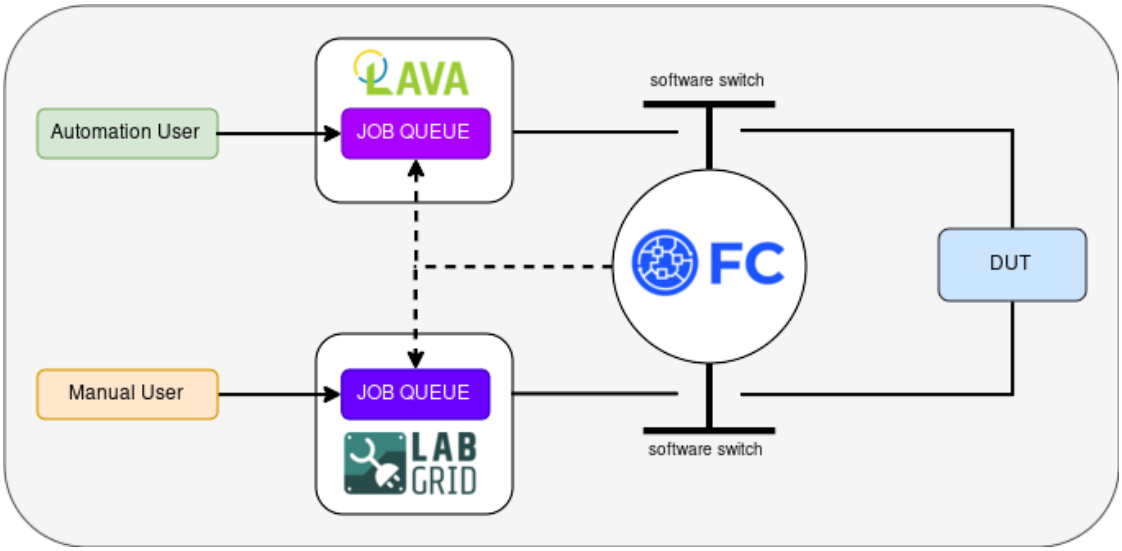
CONTENTS

4.1 Overview

4.1.1 Principle

When multiple frameworks try to access the same DUT, there are definitely conflicts there. To overcome that, we should assure only <=1 framework could get the access of DUT at the same time.

Basic



As mentioned in above diagram, FC will use different frameworks' interface to disconnect all frameworks' access to DUT by default. Then, FC will monitor the job queue of different frameworks, if find some framework's job pending on any resource, FC will connect the resource again to that framework. With that way, no conflict will be occured.

Advanced

There are two advanced features here:

- **default framework**

  FC allow admin to configure a default framework. For that default framework, FC will not disconnect its link to DUT defaultly. The DUT will be disconnected from default framework only if other framework try to access that resource, this will be automatically handled by FC coordinator.

  This is useful when some framework is treated as primary framework, then this feature could improve the schedule efficiency of primary framework. By default, all frameworks will be treated equally.

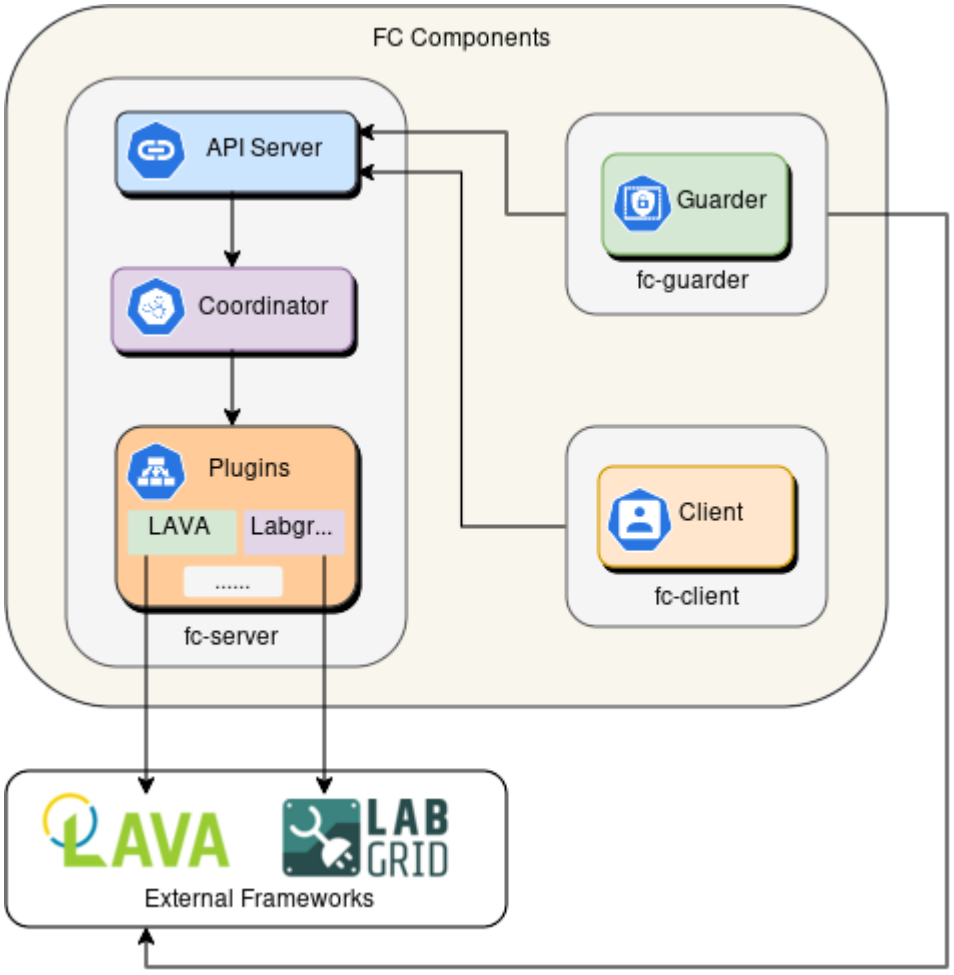
  Note: only one framework could be configured as default framework, otherwise there will be conflict.

- **resource seize**

  FC support framework priority, that means if a resource pending on a high priority framework due to an occupation of a low priority framework. The FC will force cancel the occupation of low priority framework to let high priority framework to seize the resource.

  This is useful when some framework takes critical task while other framework takes non-important task. By default, FC will use fair scheduler.

4.1.2 Architecture



See above diagram, FC has three main components inside, `fc-server`, `fc-client` and `fc-guarder`:

1. fc-server

`fc-server` is the main program to coordinate different frameworks.

- *api server:*

  There is an API server located on port 8600, it afford REST api for `fc-client` & `fc-guarder`.

- *coordinator:*

  The main component to schedule different frameworks.

- *plugins:*

  - lava:

    It will control resource by switch resource status between "GOOD" & "MAINTENANCE".

  - labgrid:

    It will control resource by inject system level labgrid reservation.

1. fc-client

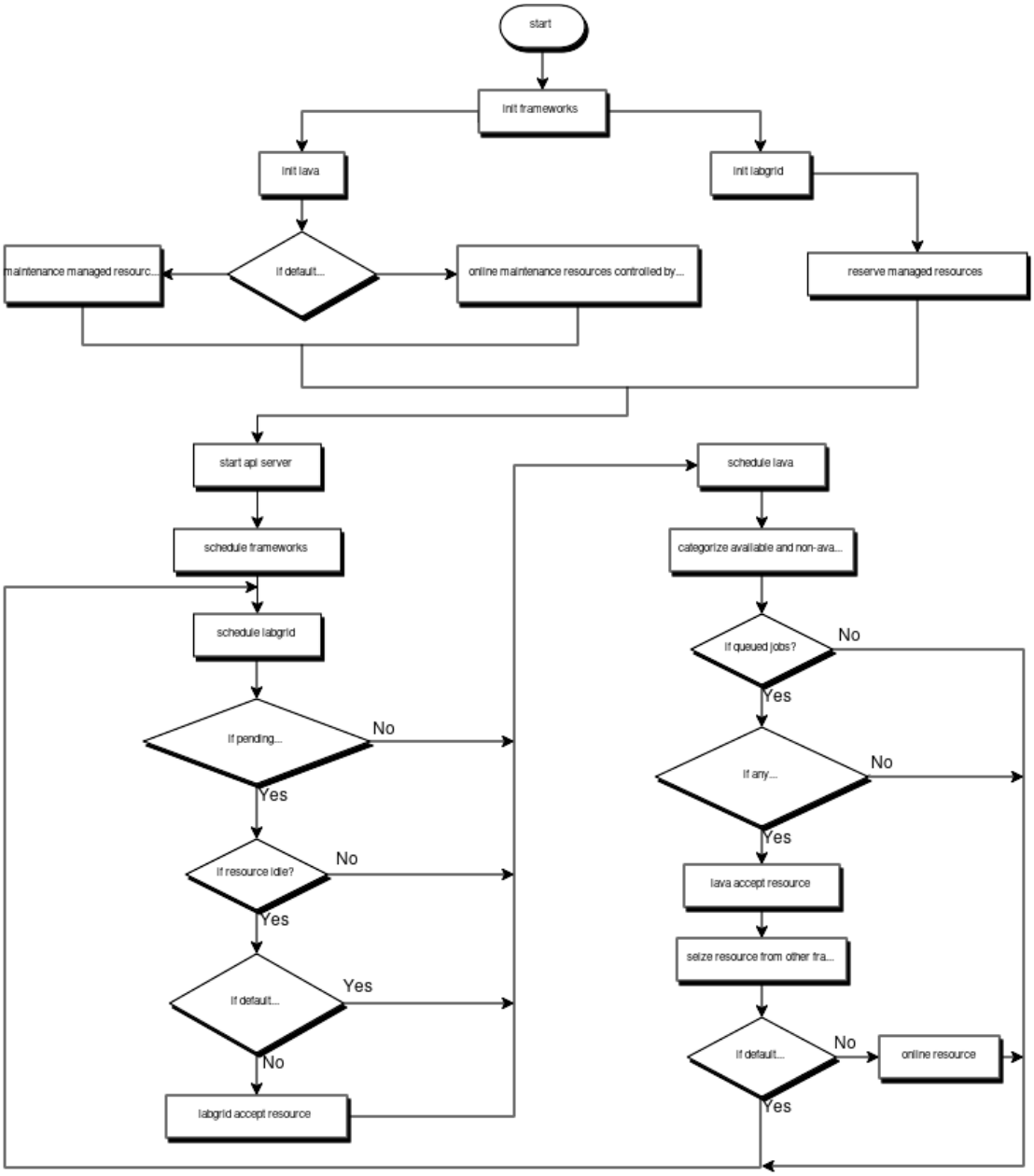
`fc-client` is the client program to query resource information from `fc-server`, meanwhile, it could help to reserve boards.

3. fc-guarder

`fc-guarder` is the guard program to monitor `fc-server`, if `fc-server` down for any reasons, the `fc-guarder` will online all lava devices to make resources still could be used by LAVA.

4.1.3 FlowChart

Next is the primary flowchart of fc:



4.2 Quickstart

4.2.1 Installation

FC could be installed by `pip`.

- **fc-server**

```
$ sudo pip3 install fc-server
```

- **fc-client**

```
$ sudo apt-get update
$ sudo apt-get install -y microcom corkscrew
$ sudo pip3 install fc-client
```

- **fc-guarder**

```
$ sudo pip3 install fc-guarder
```

Additional, you can also use FC with docker, details see *Run with docker package*.

4.2.2 Run

Note: Before run, you will certainly want to have a look for *Configuration* section to know how to configure different components.

Run with native package

- **fc-server**

```
$ fc-server
```

- **fc-client**

```
$ fc-client
```

Note: Two environments should be specified before you run `fc-client`, this let the client could find correct instance of `labgrid-coordinator` and `fc-server`.

```
export LG_CROSSBAR=ws://$labgrid_server_ip:20408/ws
export FC_SERVER=http://$fc_server_ip:8600
```

- **fc-guarder**

```
$ fc-guarder
```

Run with docker package

- **fc-server**

```
$ git clone https://github.com/NXP/fc.git
$ cd fc/docker/fc_server
$ docker-compose up -d
```

- **fc-client**

```
$ docker run --rm -it atline/fc-client /bin/bash
root@08ab13f5f363:~# fc-client
```

Note: Two environments should be specified in container before you run `fc-client`, this let the client could find correct instance of `labgrid-coordinator` and `fc-server`.

```
export LG_CROSSBAR=ws://$labgrid_server_ip:20408/ws
export FC_SERVER=http://$fc_server_ip:8600
```

- **fc-guarder**

```
$ git clone https://github.com/NXP/fc.git
$ cd fc/docker/fc_guarder
$ docker-compose up -d
```

4.3 Configuration

Three FC components require different configurations.

4.3.1 fc-server

1. fc/fc_server/config/cfg.yaml

```
registered_frameworks:
  - lava
  - labgrid

frameworks_config:
  lava:
    identities: $lava_identity
    priority: 1
    default: true
  labgrid:
    lg_crossbar: ws://$labgrid_crossbar_ip:20408/ws
    priority: 2
    seize: false

priority_scheduler: true

api_server:
  port: 8600

managed_resources:
  $farm_type:
    $device_type:
      - $resource1
      - $resource2
```

You should replace the parameters with $ symbol:

- `$lava_identity`: it's a lava concept used by `lavacli`, refers to lavacli
- `$labgrid_crossbar_ip`: it's a labgrid concept used by `labgrid`, specify labgrid exporter ip here.

- `$farm_type`: this will be shown in *fc-client* to distinguish different farm type, you could use any string
- `$device_type`: this category devices for easy readness, you could use any string
- `$resource`: list all your resources here

Some optional configure:

- `priority_scheduler`: priority scheduler only starts to work when it set as *true*
- `priority`: should specify different priorities for priority scheduler, the lower number will have high priority
- `seize`: if enable priority scheduler, all frameworks will try to seize the resource from lower priority framework, we could disable that by set *seize* as *false*
- `default`: the framework will be treated as default framework if specified as *true*

Note: The api server defaults will return `Resource`, `Farm`, `Owner`, `Comment` totally four columns to `fc-client`, but you possible to call external tool to return one more `Info` column to client.

This could be configured as next to add one `external_info_tool` to the option `api_server`:

```
api_server:
  external_info_tool: python3 /path/to/fetch_info.py $fc_farm_type $fc_resource
```

The `$fc_farm_type`, `$fc_resource` will automatically replaced by real value of resource in FC, your own `fetch_info.py` could optional to use them.

---

2. fc/fc_server/config/lavacli.yaml

You should see it in `$HOME/.config/lavacli.yaml` if you once add identities for lavacli, see this

4.3.2 fc-client

You need to define next environment variables before run `fc-client`.

```
export LG_CROSSBAR=ws://$labgrid_crossbar_ip:20408/ws
export FC_SERVER=http://$fc_sever_ip:8600
```

4.3.3 fc-guarder

You should use the same configuration with `fc-server`.

4.4 Usage

For `lava`, usage should be transparent to `FC`. For `labgrid`, the only item changed is *board reserve* with `fc-client`.

The detail usage as next:

```
$ fc-client s                                  # get all resource status managed by fc
$ fc-client -r $resource_name s                # get specified resource's status and␣
↪additional information
$ fc-client -f $farm_type s                    # get resource's status only within this␣
↪farm
```

(continues on next page)

(continued from previous page)

```
$ fc-client -d $device_type s                    # get resource's status and information␣
↪only belongs to specified device type
$ fc-client -f $farm_type -d $device_type s      # get resource's status and information␣
↪only belongs to specified device type and within this farm
$ fc-client -r $resource_name l                  # lock the resource
$ fc-client -r $resource_name u                  # unlock the resource
$ fc-client b                                    # list current bookings
$ labgrid-client -p $resource_name console       # get serial access
$ labgrid-client -p $resource_name power cycle   # power restart resource
$ labgrid-client -p $resource_name power on      # power on resource
$ labgrid-client -p $resource_name power off     # power off resource
```

Note: The additional information only be displayed when user specify resource or specify device type.

4.5 Support

Go to https://github.com/NXP/fc for help.