Sergio Alejandro Vargas
email: savargasqu+git@unal.edu.co
GitHub: @savq

# An Alternative to `Distributed` for Pluto.jl

## Introduction

Pluto uses the `Distributed` module (part of Julia's stdlib) to create worker processes and to send Julia data structures between them. These worker processes cannot create other processes themselves, meaning that —in practice— Pluto notebooks cannot use `Distributed`.

This project aims to solve this problem by replacing `Distributed` in the Pluto codebase with a new package for inter-processes communication (IPC), so that Pluto notebooks can use `Distributed` without any issues.

Implementing this solution will allow Pluto users to leverage `Distributed` for computationally intensive tasks without leaving the familiar interface of Pluto notebooks. This might make it easier to prototype distributed systems in Julia. It might also be useful in educational contexts to explain the basics of distributed computing, since Pluto is already used in various Computer Science courses.

## The Project

`Distributed` clearly distinguishes between the initial Julia process —the *master*— and other processes —the *workers*—. In particular, only the master process can create or destroy worker processes.

Pluto uses `Distributed` to execute the code of each notebook in separate environments, i.e. the Pluto server is the master process. If a user wants to use `Distributed` in a Pluto notebook, they won't be able to create new processes because the notebook is executing in a worker process.

To solve this, the project will be divided in two parts:

1. Create a package for process management and IPC.
2. Integrate said package into the Pluto codebase to remove `Distributed`

### Part I — `Distributed` Alternative

The first part of the project consists of the design and implementation of a stand-alone package to create isolated Julia processes. This package should be able to replace `Distributed` on the Pluto codebase, but it will not implement all the functionality of `Distributed`. For the rest of this proposal, this package will be referred to as **DA** (`Distributed` Alternative).

As the project is centered around inter-process communication. Some of the technical details DA will need to consider are:

**Isolation:** The main reason Pluto uses distributed processes, as opposed to threads, tasks or other asynchronous mechanism, is that processes are isolated from one another. Each process has separate memory and separate namespaces, meaning they can load separate packages, or different versions of the same package.

**Communication:** `Distributed` processes communicate by default using TCP (using the Sockets module in stdlib). DA could follow a similar design, using TCP sockets and Julia's built-in capabilities for asynchronous programming: tasks and channels.

Other alternatives that could be considered for IPC are:

- Unix domain sockets: Since DA processes will run in the same machine, they could communicate using Unix domain sockets (or named pipes on Windows). This approach could lead to faster communication between processes, but it might be limited by the cross-platform capabilities of Julia (Pluto already has some platform specific issues, see Error Handling below). It will be necessary to study the Libuv documentation (library used by Julia) and to write a prototype to determine the viability of this approach.
- ZeroMQ: Jupyter uses ZeroMQ to communicate between the Jupyter server (master) and the kernels (workers) Julia already has bindings to ZeroMQ, so this alternative could save some time as opposed to implementing the TCP (or the UDS) server from scratch.

**Serialization:** `Distributed` uses the `Serialization` module (par of the stdlib). Currently there doesn't seem to be any reason not to use `Serialization`, but if an alternative were needed, Pluto already uses the MessagePack protocol to communicate with the front-end, and that code could be re-used here.

**Error Handling:** Pluto is meant to be resilient to errors in the user's code execution (the user should be able to make mistakes!) Worker processes should handle errors appropriately and communicate them to the main process; and the master process should be able to suspend the execution of any worker process.

Pluto has existing problems with signal handling on Windows (see fonsp/Pluto.jl#452), making it difficult to terminate processes in some circumstances. It may be worth looking into whether signal handling can be properly emulated on Windows via Libuv; or if it's already being done, find why it's not working. In any case, the main goal for DA is providing the functionality that's already works with `Distributed`, so this issue will only be an optional deliverable, and it'll be addressed if the second part of the project is complete sooner than planned.

### Part II — Integration with Pluto

The second part of the project consists in integrating DA into the Pluto codebase, replacing all uses of `Distributed`. This includes:

- Porting the actual code.
- Adding a test suite that checks `Distributed` can be used correctly.
- Updating the documentation for Pluto to make clear how DA is integrated. Since DA will be a bespoke package it's specially important that future contributors to Pluto can understand it with ease.

The API of DA will be limited to the requirements of Pluto. It's not meant to be a full replacement of `Distributed`, and it's not meant to be a widely used package. As such any guarantee of API stability will depend on the requirements of Pluto.

For completeness, it's worth mentioning that the features of `Distributed` that DA *will not* implement are:

- Communication between different machines: All processes will live on the same machine as the Pluto server that spawned them.
- Different network topologies: Pluto only needs to allow communication between the main process and the worker processes (equivalent to the `:master_worker` topology of `Distributed`).

The project can be considered complete when the issue Pluto.jl#300 is closed.

### Optional deliverables

Beyond the main goal of the project, some nice-to-haves might be:

- Resolve the issue Pluto.jl#452 mentioned in Error Handling.
- Write a Pluto notebook that showcases `Distributed` or serves as a tutorial on distributed computing. One possible topic that's of personal interest is IPC using the Open Sound Control protocol. A notebook-tutorial on OSC could be very interesting, and wouldn't require much background knowledge.

## Schedule

The project can follow the regular GSoC timeline, starting in June 6 and ending in September 12. I may be unable to code during the penultimate week of the program (first week of september). However, at that point most of the coding tasks should be complete.

The breakdown of the project for the 12 weeks of the program might look like:

- W1: TCP and UDS server prototypes
- W2: Study ZeroMQ

- W3: ZeroMQ server prototype
- W4: Settle on an IPC implementation.
- W5: Buffer (in case of any delay)
- W6: Finish package and present as part of the first evaluation
- W7: Familiarize with Pluto code base
- W8-10: Integrate package with Pluto
- W11-12: Tests and documentation

## About me

I'm a Computer Science student at the Universidad Nacional de Colombia (I expect to graduate on fall 2023). I learned to program using Processing and Arduino around 2017, and I've been interested in programming learning environments since then. I wrote my first lines of Julia the day after I saw the first Pluto talk during JuliaCon 2020. (it's a silly regex exercise)

While I've been programming for years now, I still feel like I've not contributed to "large" software projects, and I'd really like to contribute more to projects that I already use regularly, like Pluto.

I've worked on smaller open source code projects. The most used program I've written is a small plugin manager for the Neovim text editor in Lua: Paq-nvim (~300 users by GitHub's estimates). It uses Neovim's libuv bindings to call file system operations, and to spawn `git` as a separate process. I think some of the things I've learned on IPC and async programming while developing Paq would be directly applicable to this project. I don't have any notable project in Julia, but I've contributed to "adjacent projects" like the Julia tree-sitter grammar.