0.1. #fix_distribution_problem

Imagine you have 100 devs split into 20 groups having 5 people working on a task. You'll have power. **Now imagine all 100 people walking in the same direction yielding 20x the power**

0.2. The driving force

TIME = MONEY Thus minimize time allowing to reuse programmers and previous investments

0.3. THE PROBLEM

There are at least 2 problems to be solved

- software distribution and orchestration

0.4. software distribution: THE PAIN

Its absolutely impossible to create a README.md file on github and have something which works for all users. So you often find the many installations guides. But people want copy paste and be done

0.5. software distribution: THE SOLUTION

Create a cross language and platform way to describe

- dependencies
- ABIs
- #platforms Section 0.5.1 (to keep simple things simple)

and target multiple distribution systems:

- OS + install manually what doesn't work
- nix
- brew
- ....

0.5.1. #platforms

It often makes sense to settle on some major versions of many packages to keep the load low. Eg Haskell has platform to guarantee some libaries work together of that platform version.

0.5.2. (simplified) example usage

pool = [this system, that repository] solve_pool(pool, for = “firefox”)

Which basically reads as use os provided packages (eg apt get or brew on osx) but if not found build from scratch adding nix like guarantees

0.6. Other solutions partially failing

0.6.1. Why Nixpkgs fails

Well it suuceeded. Claiming to be biggest linux distro out there. Yet its very hard to estimate the time it requires to package something new beacuse some dependencies are burried in cmake files and keeping this all up to date and moving into nix space is error prone and time consuming.

The better way is to create a new abstraction which then can target cmake and nix.

0.6.2. Why C# fails ?

F# is 30-50% less code No cross plattform GPU abstraction such as Julia ?

0.6.3. Why F# fails ?

Same as C# ?

0.6.4. Why brew partially fails

If you need expermiental code its hidden behind let add this or that repository or requiring special version. That’s not easily automatable.

0.6.5. Why Gentoo fails

Can’t compile on multiple machines like Nix

0.6.6. Why Rust’s Cargo partially fails

Well much better than C++, but if you depend on C++ you have to make cargo build it. So no more sharing.

0.7. Story GPU

with that hosting company having strong GPU do machine learning, yielding a future result in a distributed store which then can be accessed on multiple devices for inference.

0.8. Story rust cross compiling to windows

Rust cross compilation on OSX requires installing docker and fetching round about 1GB of data.

By syncing code to server using homeless-and-derived-contents the code even can be created in the browser within seconds.

0.9. ROADMAP

- create pools of software for existing distributions windows: chocolately, scoop, win-get osx: brew, nixpkgs linux: nixpkgs, ubuntu, ….
- create pools of source and dependencies
- allow to mix, eg mixin pool of current distro, then augment with source repositories, then solve and do magic
- target different distribution systems rpm/pkg nixpkgs gentoo conda brew …

- development and distribution bazel buck2
- homeless homeless will be able to pick up sources and conversions thus will be able to distirbute future results with an execution and distribution engine eventually finding the best path

0.10. TODO

Harmonize with homeless:

The concept of derived data is the same here and with homeless. Eg assembling a video is conceptually the same with assembling an Android app.

0.11. glossary

0.12. references

ODO there is so much missing here