Assessing future software options

Jim Pivarski

Princeton – DIANA

July 25, 2016

Terminology

Hallway warning:

"Even if software X is better, physicists won't use it because it's unfamiliar."

- Supported by anecdotal evidence. Can we get systematic?

The Cuisinart problem:

It slices and dices, but do you want to set it up and clean it?

- Physicists regularly have to make pragmatic decisions about whether the overhead in using a tool is worthwhile.

Top-down versus bottom-up:

Collaboration frameworks, filesystems, and cluster infrastructures get decided upon by a few people and used by many.

End-user analysis tools get evaluated by each user.

Sociology of physicists

I'm planning a focus group to study physicists' opinions on

- relative importance of familiar syntax
- relative importance of computational speed
- relative importance of collaborators using same tools
- relative importance of size of community (in and beyond HEP)
- willingness to try advanced techniques: analysis in a cloud, functional chains, metaprogramming
- possibly with Spark/Julia examples to make things concrete

The format is

- closed/anonymous, but producing a public statement by concensus
- limited to physicsts actively typing analysis code (mostly students/postdocs); about ten participants

This is a starting point

I've been reading up on focus groups; usually they should be repeated a few times to find out if you've fully sampled the space of opinions or if dominant voices biased one of the groups.

Moreover, I have no experience in this form of research. Would we be able to attract interest from social scientists who can do this kind of study professionally?

Focus groups are often the starting point with surveys as a follow-up. (The open-ended focus groups determine what questions should go on the survey and how to phrase them.)

Might we analyze HEP software users in a routine way with regular, scientific surveys?

Backup: summary of Julia talks

(including points not raised)

- Julia uniquely "feels like Python" and "runs as fast as C."
- Has built-in multiprocessing through ssh, like a lightweight PROOF cluster. (Killer app not mentioned???)
- Calls out to C, C++, Python, R, and Java.
- Has quirks like 1-based indexing and lack of OOP-like methods: `event.track(2).hit(55)` → `hit(track(event, 2), 55)`
- Lack of compile-time safety could be bridged by reflection.
- Demonstrated interface with ROOT, but issues with conflicting LLVM in Cling.
- Expects major additions before version 1.0 (e.g. threading).
- Many packages are in early stages of development.
- Relatively small community outside HEP, compared to Hadoop/Spark, Scikit-Learn, or R.