What aren't we talking about that we should?

Stan Seibert
2019-10-28

What aren't we talking about that we

What software stuff do I hope you are talking about?

Who is this guy?

- Did particle (astro)physics experiment until 2013
- Worked on: SNO, Braidwood, SNO+, MiniCLEAN, & LBNE (before it was DUNE)
- Wrote RAT (simulation / analysis framework) for Braidwood, refurbished for MiniCLEAN, adopted by SNO+
- Evangelized GPU computing starting in 2007 to anyone who would listen
- Went into industry, landing shortly at Continuum Analytics, now called Anaconda
- Currently manage the open source development team at Anaconda, where we work on **lots** of Python projects
- *Note: Not wearing my Anaconda hat for this talk. These opinions are my own.*

Format

- No idea what is common knowledge in the field after 6 years
- Going to throw out some "hot takes" that I mostly believe 😄
- Hopefully will spark some ideas or interesting conversation
- Apologies if these points are obvious

Idea #1: Docker *might* be overrated

- Docker is amazing for creating well-defined Linux compute environments
- Huge advance over previous status quo to see more reproducible science and production data processing done with Docker
- But...
  - Docker was designed for microservices where composition happens at the network layer
    - Poor abstraction for HPC
    - How do you combine container A and container B? Swizzle Docker files?
  - Docker can be used to hide / excuse poor packaging and portability practices
  - Docker has a crazy security model that is incompatible with traditional multi-user systems
- *Keep looking for better ways to do this! Check out other container options, like Singularity, and don't give up on user-space packaging as complementary capability to containers.*

Idea #2: Accelerated Computing is Going to Fragment, Hard

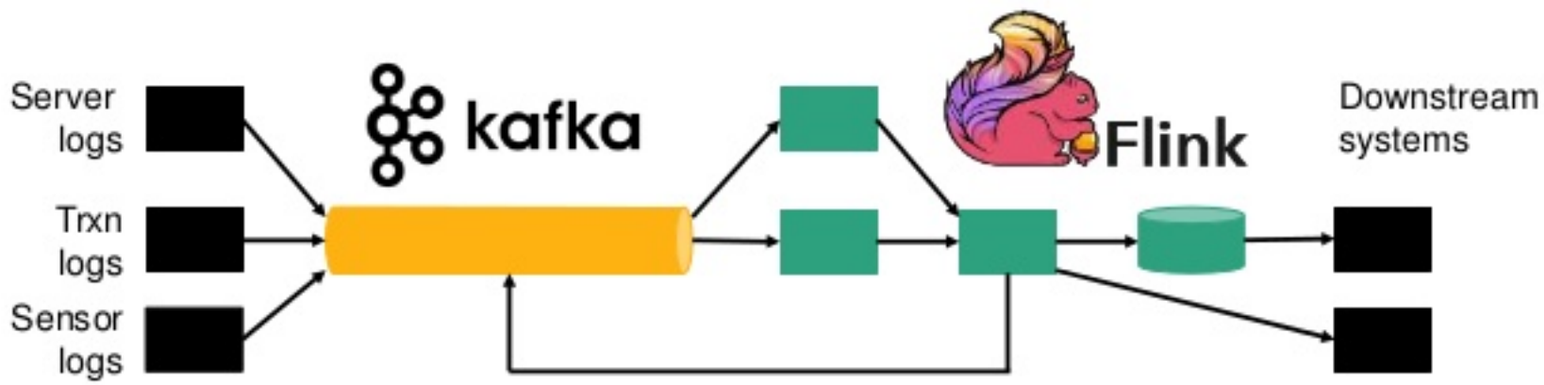
- The multicore x86-64 monoculture died years ago, but it is taking time for that shockwave to propagate outward:
  - CUDA was the first viable fragment, and we still haven't figured out how to fully exploit it.
  - CPUs now have GPU-like hyperthreading and SIMD instructions
  - PPC (via POWER8/9) is now a viable HPC contender
  - ARM is rapidly growing into the server space
  - AMD GPUs are behind the curve (ecosystem-wise), but growing
  - Intel is releasing a GPU in 2020
  - FPGA makers are taking the "numerical coprocessor" role more seriously
  - ML hype means everyone is trying to make a matrix coprocessor now
- OpenCL basically failed to unify all this (though it may rise again), so don't wait for that.
- *Don't look for a bandwagon to jump on, but be open (and prepared) to take advantage of unusual compute platforms for DAQ and/or event processing.*
  Where are your bottlenecks?
  How hard is it to move data in and out of your analysis frameworks?

Idea #3: Data Layouts Matter More than Algorithms

- Why aren't we able to all this novel compute hardware (or even the AVX instructions)??
- Our data has been sacrificed on the altar of C++ OOP
  - Objects full of pointers to heap allocations of objects with yet more pointers...
  - Bad for caches, bad for SIMD, and near impossible to relocate from CPU to an accelerator with a distinct memory space
- ROOT partially saves us from ourselves due to how TTree's are basket-ed in files.
- *Position-independent, columnar storage is going to continue to be important to take advantage of new technology. Apache Arrow has good ideas, and Awkward Array is very promising.*

Idea #4: Can we take ideas from Stream Processing frameworks?

- Data engineers in industry now have to deal with streaming data on par with (some) running physics detectors
- In physics, both online monitoring and offline data processing frameworks deal in transformations and aggregations on sequences of events
- Tend to be very Java-centric
- Maybe there are API and code organization ideas to be borrowed from projects like Apache Kafka and Flink?



https://www.slideshare.net/stephanewen1/apache-flink-overview-at

Idea #5: Interactive Computing at Scale Requires Going Beyond Batch Schedulers

- Notebooks are great (for some things), and they make interactive computing very attractive
- Interactive data analysis and exploration should operate on human time scales
  - Ideally we want the human to wait less than O(10 sec)
  - Compute may need to idle for O(seconds or minutes) for human to decide next step
- Nominal utilization on HPC clusters will seem high (many job slots filled)
- Actual utilization will be very low (small average CPU load)
- Rapid autoscaling of compute resources will be essential
- Kubernetes heads in the right direction, but may not go far enough
- *Check out what Pangeo project is doing with Dask and Kubernetes for geoscience*

Idea #6: Turn Your Simulation / Event Processing Frameworks Inside Out

- (Here, I hope state of the art has changed since my time)
- Many frameworks I worked on were monolithic apps with their own control language for loading events in a loop and pushing them through a sequence (sometime branching) of event processors.
- Consider abandoning these bespoke scripting DSLs for a general language (like Python) and make the framework a true library with reusable components
- Jobs can combine components however they like, using standard control flow
- Makes it easier for components to depend on each other (fitters that call into geometry routines, etc)
- True DAG workflows are easier to describe with a general purpose programming language.
- Also forces you away from relying on massive (and error-prone) hidden global state once you don't have an "app" to manage it all.
- ***More Lego, less player-piano***

Idea #7: Software Testing in Physics is Underappreciated

- Software used by more than 1 person (*[you]* & *[you + 3 months]* are different people) needs automated testing.
- Software developers fight over unit testing vs. integration testing vs. acceptance testing which is both useful and meaningless posturing. Don't get sucked into the taxonomy debates!
- Acknowledge the utility, but be skeptical of popular testing techniques:
    - **Reproducible analyses:** Can my colleagues get the same wrong answer this week that I got last week? See also: "Golden files" which you must match after code changes.
    - **Error Correcting Graduate Students:** N graduate students write their own analyses mostly from scratch and we make them duel each other until they all agree.
    - **Oracle Theses:** Past students document their results, and you keep fussing with your implementation until you reproduce their summary tables or plots.
- *How do we make testing generation easier and less onerous for busy people?*
- Testing numerical code is a distinct skill that is frequently not addressed in software engineering tutorials
- When should tests be statistical vs. exact? (Fixing your RNG seed seems like a good idea, but is really kicking your test brittleness down the road.)
- *Take a look at Hypothesis for interesting ideas on finding corner cases by automatically generating tests?*