Frameworks @ EIC Software Workshop

- FairRoot (GSI, 2004), based on AliRoot (ALICE @ CERN-LHC, 1998)
- art (FNAL, 2009), based on CMS software (CMS @ CERN-LHC)
- JANA (Hall D, 2004), in-house
- Fun4All (PHENIX/sPHENIX @ BNL, 1998), in-house
- CLARA (Hall B), loosely based on GAUDI (LHCb @ CERN-LHC) (talk was canceled)

Framework Aspects

- Computing model (node, batch, cluster, grid)
- Degree of ROOT integration
- Support for concurrency/multithreading
- Extent of predefined workflows
- Support level
- Pre-existing work

Things I Learned (I)

- Outside of JLab, most NP and HEP experiments use ROOT for object serialization and file I/O. Oddly, at JLab, there are efforts to do this without ROOT.
- ROOT is indispensable for final interactive analysis
- Interactive steering is not always the best choice
  - Configuration files can be more readable than scripts
  - Mostly useful for testing and debugging
- New developments integrate the simulation into the framework
- Everyone does multi-stage analysis with intermediary DST files
- Run-time configuration (no recompilation) is standard
- EIC group has made good progress with setting up and end-to-end analysis chain under FairRoot

Things I Learned (II)

- The case for concurrency may be less compelling than I thought
  - Memory usage and I/O performance arguments are no longer as strong as they used to be
  - Job schdulers achieve similar results without the programming hassles for the physicists
  - Serious data challenges must be addressed with distributed computing (cluster or grid) anyway

SoLID Data Parameters

Experiment	Event size (kB)	Trigger rate (kHz)	Data rate (MB/s)	Raw data (PB)
SIDIS	3	100	300	5.6
PVDIS	50	20	1,000 $\xrightarrow{\text{HLT}}$ 300	7.0
cf. GlueX	15	200	3,000 $\xrightarrow{\text{HLT}}$ 300	3.2/yr

Choosing A Computing Model

3 minute run → 18M SIDIS events, 50 GB raw data
Assume 20 ms/event → to keep up with 100 kHz event rate, need 2000 cores

- **Single-threaded:** no framework support for parallelism
  - 2000 runs in parallel → 100 TB disk space for input
  - ≈ 100 hours turn-around time per run
  - Problems: inefficient in cost & turnaround time
- **Multi-process:** parallelism through external job scheduler
  - *E.g.* 32 single-threaded jobs working on different event ranges of one run
  - 62.5 runs in parallel → 3 TB disk space for input, 3 hours/run
  - Potential problems: I/O bottlenecks (disk head thrashing), limited scalability, complexity outsourced to job scheduler
- **Multi-threaded:** event-level parallelism built into software architecture
  - Similar to multi-process, but reduced random disk access & memory footprint
  - Problems: scalability limited by cores/node, code complexity
- **Distributed (cluster, grid):** event-level parallelism through built-in scheduler
  - 1 run in real time, 0.05 TB disk space for input.
  - Virtually unlimited scalability
  - Potential problems: even more complexity, network & tape I/O bottlenecks

My Take On the Computing Model Choice

- A multi-threaded design offers
  - best performance in terms of I/O and memory use
  - reasonable compromise in terms of complexity
  - sufficient scalability for SoLID needs
- A distributed system can be built on top of a multi-threaded implementation