SoLID Simulation & Reconstruction Software

Ole Hansen

Jefferson Lab

SoLID Collaboration Meeting
June 30, 2017

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 $\overset{\text{HLT}}{\rightarrow}$ 300	7.0
cf. GlueX	15	200	3,000 $\overset{\text{HLT}}{\rightarrow}$ 300	3.2/yr

SoLID in Context

- Computing requirements comparable to Hall B & D's
  - "Solved Problem" by the time SoLID runs
  - No truly new requirements on software. Hall B/D's software would do, as well as anything else similarly designed

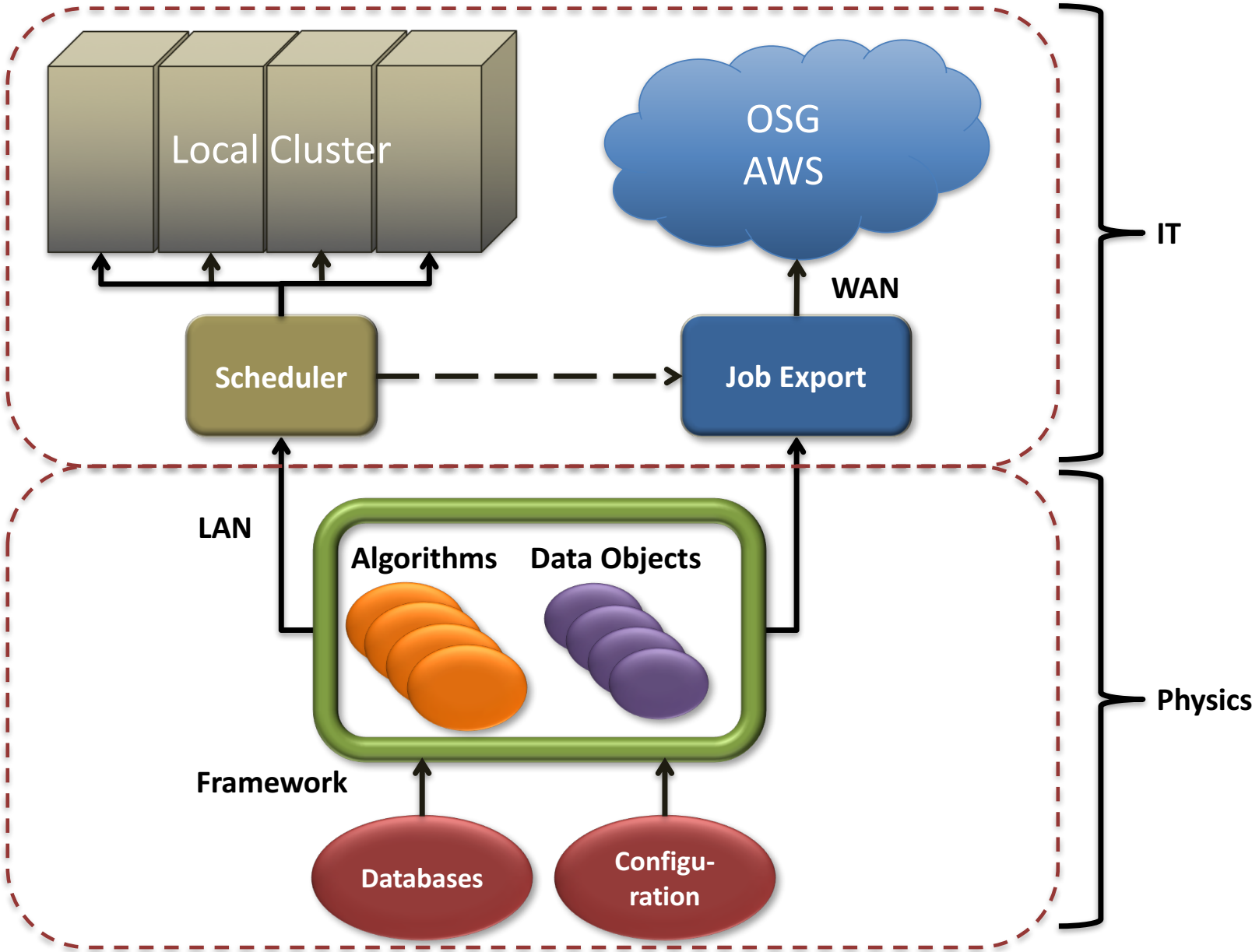
Processing Petabyte-Size Data Sets

- Physics software aspects
  - File format must scale → proven solution: ROOT
  - Parallelization/multi-threading desirable, especially for on-site datacenter processing (our "farm")
- IT infrastructure factors
  - Data access speed matters greatly → tape, disk, file system, network
  - Must have enough CPUs → $$$
  - Carefully choose computing model

Computing Model: Distributed, Cloud, Grid, Cray, etc.

- Distributed (cluster) computing and grid/cloud-friendliness increasingly important to provide "elasticity" for peak demands
  - Does not usually require specially written sim/reco software
  - "Super-framework" (scheduler, orchestrator) for distribution
  - Deployment tools for grid/cloud
  - Grid/cloud unrealistic for reconstruction (network is bottleneck), though this could change
- Simulation & Reconstruction software **does not** generally run on supercomputers. Theory codes do (including event generators). Requires specially written code.

Physics Software and Computing Models



Software Components We Need

Component	Main subcomponents	Packages / Starting points	Time required
Databases	• Geometry • Conditions / Calibrations • Run information, run list	• TGeo, (DD4hep?), GDML • CCDB (Hall B/D) • (RCDB?) (Hall B)	Moderate
Data model / structures	• Digits, Hits, Clusters • Track candidates, Tracks • Particles	• Hall B/D • other large experiments	Low
Algorithms	• Event Generators ✎ • Digitization ✎ • Cluster finding • Track finding / fitting ✎ • PID	• Std MC generators (Pythia etc.) • Existing KF codes (various) • Genfit (track fitting) • Hall B/D	Very High
Decoder	• EVIO interface • Event reassembly • Pipelined electronics processors • Mapping tables / database	• Hall A/C decoder • DAQ group • Other halls?	Moderate
Event display	• Desktop viewer • (Web viewer)	TEve (ROOT)	Low / Moderate (exists)
Object I/O		TFile / TTree (ROOT)	Low (exists)
Processing Framework	• APIs (algorithms, data, services) • Job configuration • Event loop • Parallelization (event/task level)	Various (see later)	Low (if reusing) High (if custom)
Software management	• Repository ✓ • Bug tracker ✓ • Build system, platform support • Packaging	• GitHub • Redmine • CMake • RPM, (Spack ?)	Low

Software Effort Estimate

https://hallaweb.jlab.org/12GeV/SoLID/download/doc/Estimated_SoLID_Offline_Effort.xlsx

26	Digitization testing		20	
27	DAQ/Trigger emulation	JLab	16	
28	Framework integration	JLab	8	
29	Code testing/QA		6	
30	Activities coordination	Duke	12	
31				----------
32	Subtotal Simulations			144
33				
34	**Reconstruction**			
35	Framework	JLab		
36	Build system		3	
37	ROOT tree output module		6	
38	Multi-threading		12	
39	Distributed architecture		12	
40	Documentation		16	
41	Database API & integration	JLab		
42	Geometry		4	
43	Conditions		4	
44	Mapping		2	
45	Requirements determination	JLab		
46	Data model	JLab	4	
47	Tracking	Duke, JLab		
48	GEM cluster analysis		6	
49	GEM track finding SIDIS, J/Psi		24	
50	GEM track finding PVDIS		12	
51	GEM track fitting		8	
52	Calorimeter cluster finding	UVa, JLab	8	
53	Cherenkov amplitude analysis		2	
54	MRPC ToF analysis		2	
55	Overall PID framework	JLab	4	
56	Event display	JLab	8	
57	EVIO raw data decoder	JLab	12	
58	Slow Controls integration	JLab	16	
59	Testing/QA	JLab	36	
60	Activities coordination	JLab	12	
61				----------
62	Subtotal Reconstruction			213
63				

- 25% contingency
- 75% "developer efficiency"

98			----------
99	Sum		586
100			
101	Contingency	25%	147
102			----------
103	Sum with contingency		733
104	Developer efficiency	75%	
105	(overhead for training		
106	and collecting requirements)		
107	Overhead from efficiency		244
108			----------
109	Total estimated effort		977
110			

Total: 977 FTE-weeks!

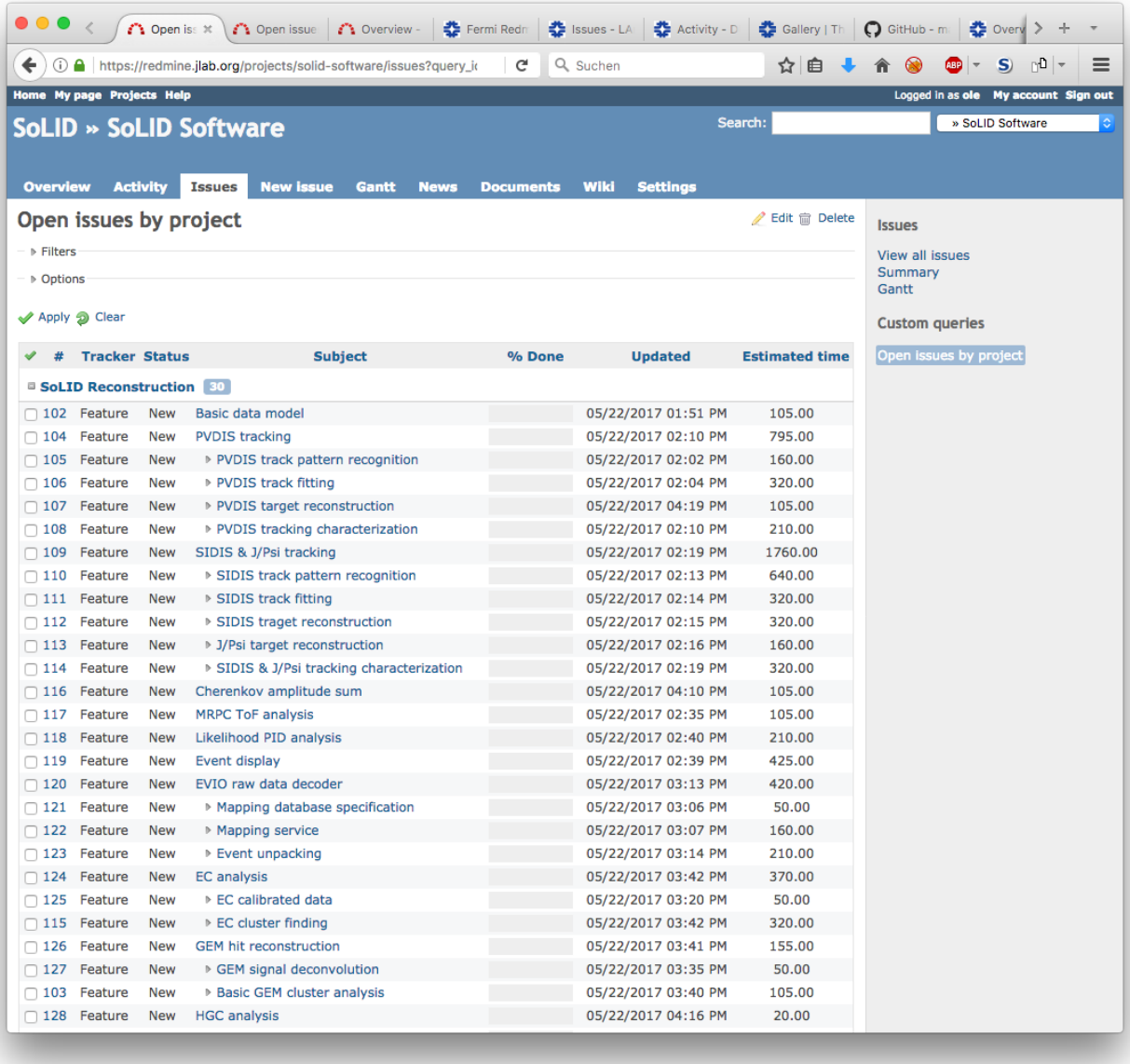
Comparison with GlueX Software Effort Estimate

Task Group	Labor estimate (FTE-weeks) GlueX	SoLID	Main reasons for difference w/GlueX
Simulation	192	240	Simulation to be integrated in framework.
Reconstruction	787	355	Adoption of an existing framework. Smaller number of subsystems. Re-use of algorithms.
Calibration	275	104	Smaller number of subsystems.
Production	275	155	Standard data format. Re-use of JLab workflow tools.
Analysis	275	100	No PWA analysis and no grid implementation of analysis.
Data Challenges	62	23	No PWA data challenge.
Totals	1866	977	= 22 FTE-years

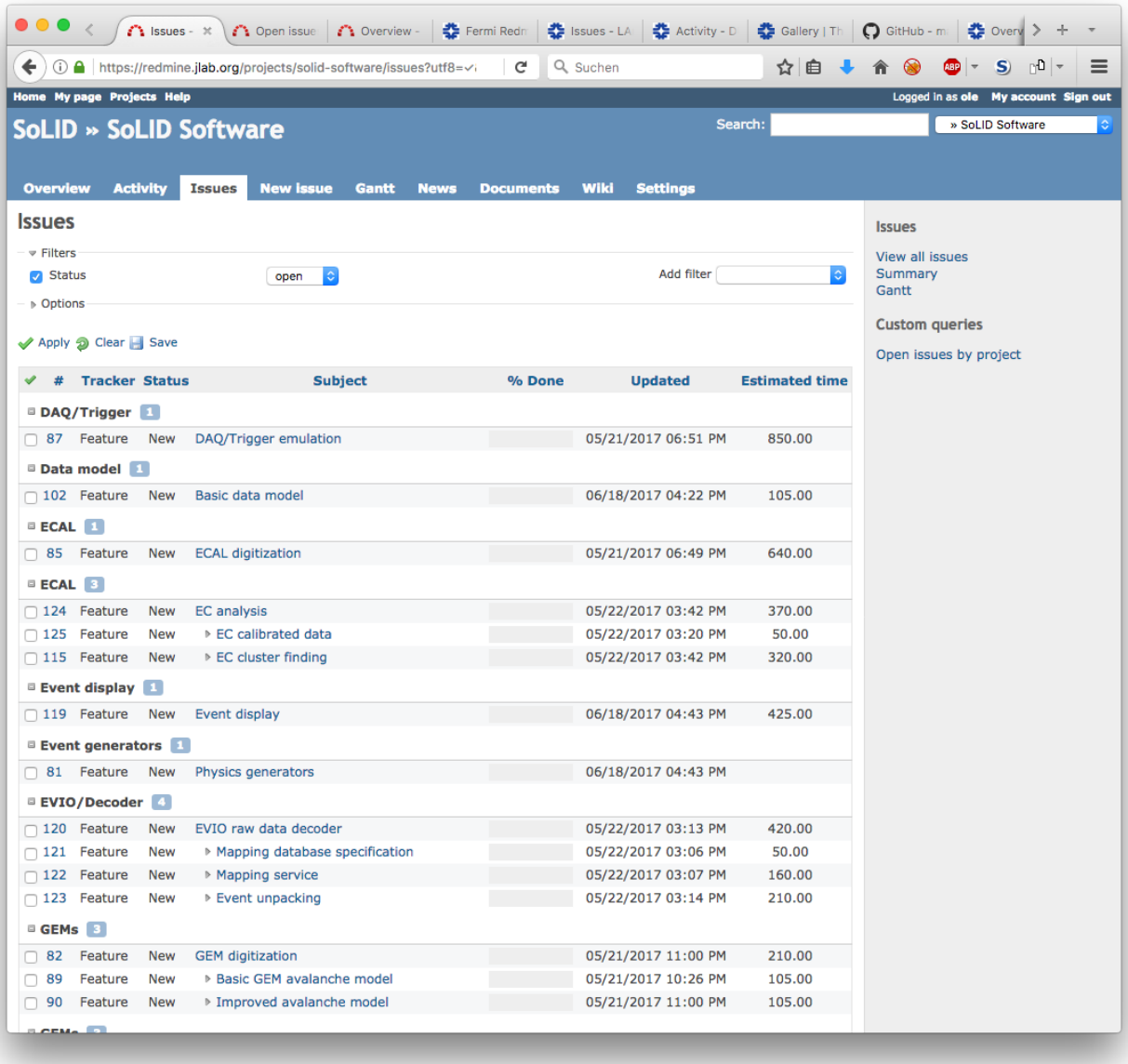
Available FTEs

- 22 FTE-years = 4 developers working full time for 5.5 years
  - **Imperative to start NOW!**
- Currently have 4–5 users/staff working at $\leq 50\% = \leq 2$ FTEs
  - Severe FTE shortage
  - Try to improve: tap collaborators, get funds for new hires ...
  - Reuse existing software components as much as possible!
  - Avoid detours, delays, "nice to have" items. "Good enough" will do.
- Full time/long term developers more effective than part time personnel!
- Good project management will help, too

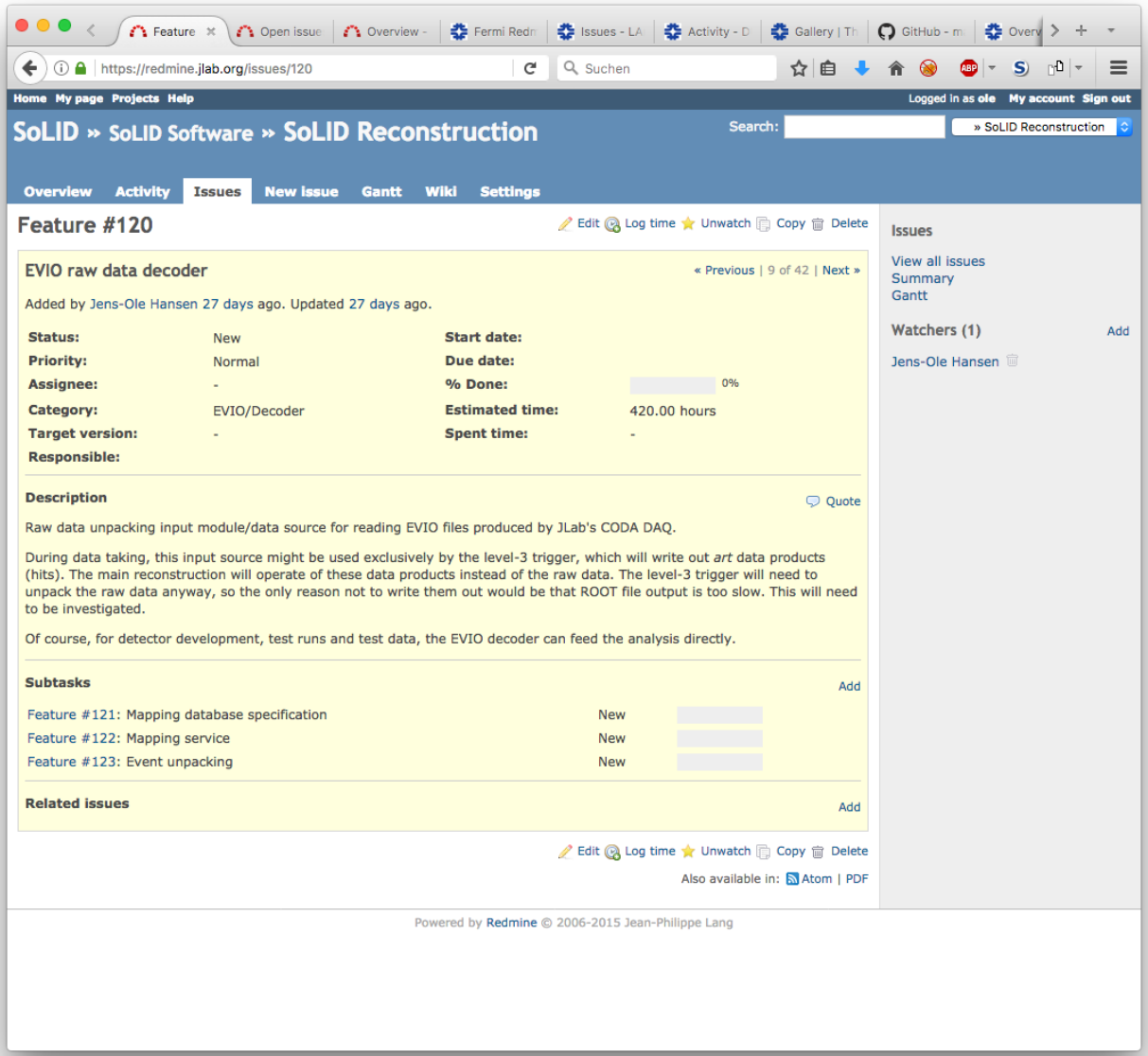
Redmine Issue Tracker `https://redmine.jlab.org`



Redmine Issue Tracker — Issues by Subsystem



Redmine Issue Tracker — Issue View



Reusing Software Components

- Geometry handling
  - Ideally describe geometry in terms of high-level "core parameters" plus code
  - DD4hep looks promising, but it's at best beta quality
  - In-memory representations: ROOT's `TGeo`, Geant4's
  - File formats: GDML (not necessarily needed?)
- Conditions databases
  - Hall D's `CCDB` is suitable. Adopted by Hall B, considered by Hall C
  - Run conditions package RCDB (Hall B)
- Tracking
  - Already developed own Kalman filter-based tracking (Weizhi)
  - Many packages available (e.g. `genfit`), should use for improvement
- PID
  - Similarly, many implementations available
  - Good problem for machine learning algorithms
- Event display
  - `TEve` (ROOT) should allow rapid development

Event-Processing Framework

- Standardizes access: API (Application Programming Interface)
  - Event store
  - Databases (e.g. geometry, conditions, configuration)
  - Services (e.g. histogramming, messages)
- Configures jobs
- Implements event loop
- Ideally, provides persistency I/O (data serialization)
- Frameworks tend to be purely technical. No Physics Here

Framework: Our High-Level Requirements/Specifications

- End-to-end: Framework should support all of simulation, digitization, reconstruction and physics analysis. HLT yes, DAQ no.
- Multi-pass processing: output $\rightarrow$ input for next pass
- Run-time configurable:
  - No recompilation for different analysis workflows/parameters
  - Multiple instances of modules (with different configurations)
- Multiple analysis chains per job, *e.g.*
  - Different tracking or PID schemes
  - Several physics analyses in parallel
- Extensive metadata in DSTs, *e.g.*
  - Database parameters from previous stages (geometry etc.)
  - Data provenance
- Interactive analysis with ROOT

Overall Most Convincing: *art* Framework (Fermilab)

- General
  - Based on CMSSW from CMS (LHC), forked in 2008
  - End-to-end solution, intended for simulation, reconstruction and analysis
  - Specifically intended to be a "common infrastructure" component
  - Used by FNAL "Intensity Frontier" experiments (DUNE, NOvA, Darkside50, mu2e . . .) Ca. 10 collaborations, many with data sizes $\geq$ SoLID
  - Supported by FNAL Computing Division (Scientific Software Applications)
  - Good documentation. Easy to get started
  - Likely to be long-lived due to heavy commitments by major collaborations
- Technical
  - Written in C++11/14 by experts. High code quality.
  - ROOT object I/O
  - Clear, user-friendly JSON-like configuration language (FHiCL)
  - No parallelization, but multi-threading in development, see `https://cdcvs.fnal.gov/redmine/issues/15372`
  - Input format, databases, event display and simulation engine defined by user
  - `http://art.fnal.gov`

art FHiCL configuration file

```
#include "fcl/minimalMessageService.fcl"

process_name : reconstruction

services : {
   message : @local::default_message
}

source : {
   module_type : FriendlyRootInput
   fileNames : [ "simulation.root" ]
   friendFileNames: [ "digitization.root" ]
}

outputs : {
   rootOut : {
      module_type : RootOutput
      fileName : "reconstruction.root"
   }
}

physics : {
   producers : {
      idealTracker : {
         module_type : IdealTrackingCode // Ideal hit-to-track association
         input : FWDGT                   // Consider only FWDGT clusters
         momentumSmearing : 0.1          // 10% momentum smearing
         vertexSmearing: [ 0.1, 0.1, 0.1 ]  // Vertex position smearing
      }
      recoKalman : {
         module_type : RecoKalman        // Kalman track fitter
         input : idealTracker            // using idealTracker clusters
      }
      pidCorrelator : {
         module_type : PidCorrelator
      }
   }
   reco_chain : [ idealTracker, recoKalman, pidCorrelator ]
   output_to_file : [ rootOut ]

   trigger_paths : [ reco_chain ]
   end_paths : [ output_to_file ]
}
```

Hall B Software

- In-house development
- Framework: CLARA
  - Very general SaaS architecture, format-agnostic
  - Multi-threaded and distributed
  - Custom configuration language
  - Written in Java
  - Services (algorithms) can be Java, C++ or Python
  - `https://claraweb.jlab.org`
- Application: CLAS12 Offline Software / Coatjava
  - Written in Java
  - EVIO decoder
  - In-house file format (HIPO), high-performance
  - No object I/O. Data in "bank" structures (named 1-d arrays, no nesting). Access by bank/variable name.
  - In-house geometry package, derives geometry from "core parameters"
  - CCDB/RCDB conditions database
  - Simulation is separate package (GEMC)
  - Limited documentation. Hard to get started
  - `http://clasweb.jlab.org/clas12offline/docs/software/html`

Frameworks: Technical Comparison

Feature	art (FNAL)	CLARA/Hall B	JANA/Hall D
Origin	CMSSW (CMS)	In-house	In-house
First release	2009	late 2000s?	2005
Collaborations using framework	10	1	1
Language	C++11/14	Java (C++, Python)	C++98
Output, object persistency	ROOT	HIPO (binary) (flat arrays only)	HDDM (XML)
Steering, configuration	JSON-like (FHiCL)	text files (custom syntax)	command line & compiled in
Reusable/multi-instance modules	yes	yes	very limited
Multiple analysis chains	yes	yes	very limited
Data product identification	type + 3 keys	bank name	type + tag
Complexity of data object search	O(logN)	O(1)?	O(M>N)
Data provenance tracking	yes	no	no
Test/filter modules	yes	yes	output module
Parallelism	no (MT underway)	MT + distributed!	MT (partial)
Main dependencies	cet-is (3.5 GB) (ROOT, boost etc.)	JVM	Xerces XML
Preferred installation	Binary via UPD	Source (GitHub)	Source (GitHub)
Unit tests	425	29 (CLARA)	0
User documentation	User Guide (500p), workshops	Examples, brief docs (incomplete)	Examples, Wiki, User Guide (old)

Framework Adoption Scenarios I

art

- Pros
  - Ready now
  - Wide adoption
  - End-to-end
  - Geant4 integration demonstrated
  - Good documentation $\rightarrow$ low need for support
- Cons
  - Single-threaded (multi-threading under development)
  - Custom build system
  - Concern about long-term Fermilab support (?)

Framework Adoption Scenarios II

Hall B (CLARA+CLAS12+Coatjava)

- Pros
  - Distributed & multi-threaded
  - Can keep using GEMC
  - Geometry and conditions services ready
- Cons
  - Java-based (resistance from collaborators, counter to HEP trends, incompatibility with existing libraries)
  - Cumbersome bank-based data model
  - In-house DST format
  - Simulation separate from framework
  - Thin documentation $\rightarrow$ high support burden on Hall B

Framework Adoption Scenarios III

CLARA alone

- Pros
  - Distributed & multi-threaded
- Cons
  - Little or no usable C++-based service implementations available
  - Need implement bulk of framework from scratch
- Options
  - Marry *art* and CLARA
  - Write distribution framework for *art*. Maybe less work than writing a pile of services for CLARA.

Framework Adoption Scenarios IV

Hall D (JANA/DANA)

- Pros
  - Multi-threaded
  - Lightweight (few dependencies)
  - Easier to learn than *art*
- Cons
  - Fewer capabilities, more limitations than *art*
  - Largely compiled-in analysis chain configuration
  - In-house DST format
  - Simulation separate from framework

Framework Adoption Scenarios V

EIC

- Pros
  - None
- Cons
  - Design not settled. Complete lack of specifications.
  - Vaporware

Collaboration with EIC?

- Pros
  - Common components $\rightarrow$ reduced effort
  - Easier transition between projects
- Cons / Concerns
  - Delays due to development of common specifications (possibly years)

Conclusions

- SoLID Simulation & Reconstruction software development will be a major effort to which development resources should be allocated *in the near future* to ensure readiness.
- Due to very limited manpower, we can only take a pragmatic, effort-minimizing approach with little room for experiments.
- We plan to use existing, proven resources to the maximum extent possible.
- A detailed development plan exists. We are ready to start.

Backup

State Of the Art Architecture: GAUDI Design

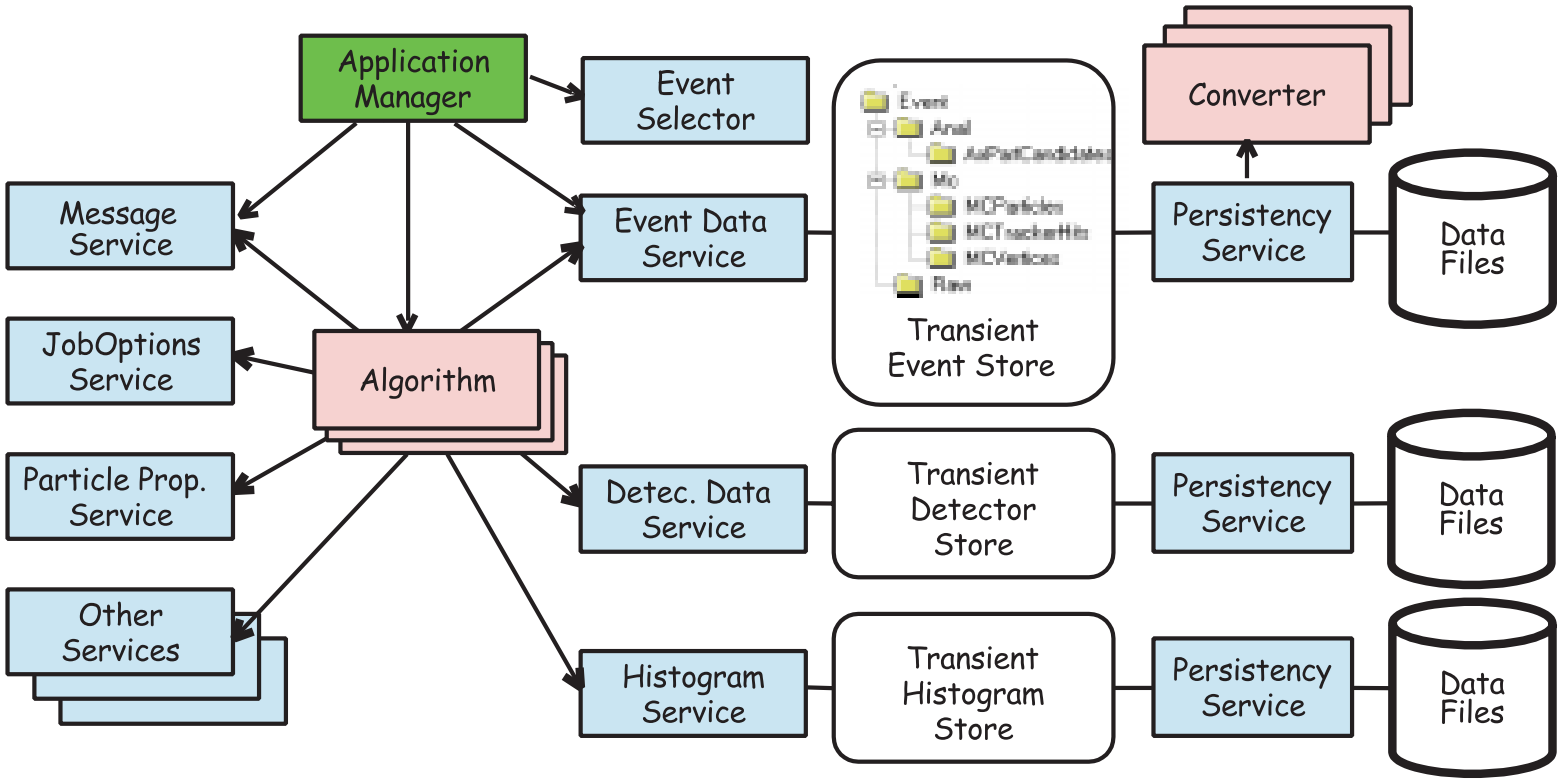
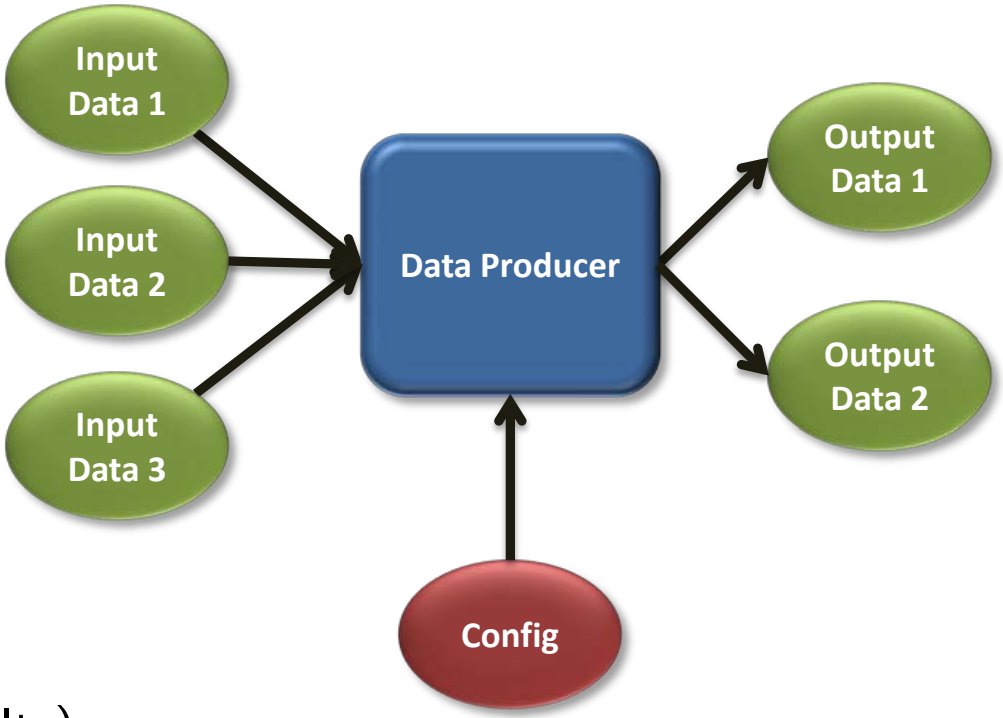


Figure 2: Object Diagram of the GAUDI Architecture

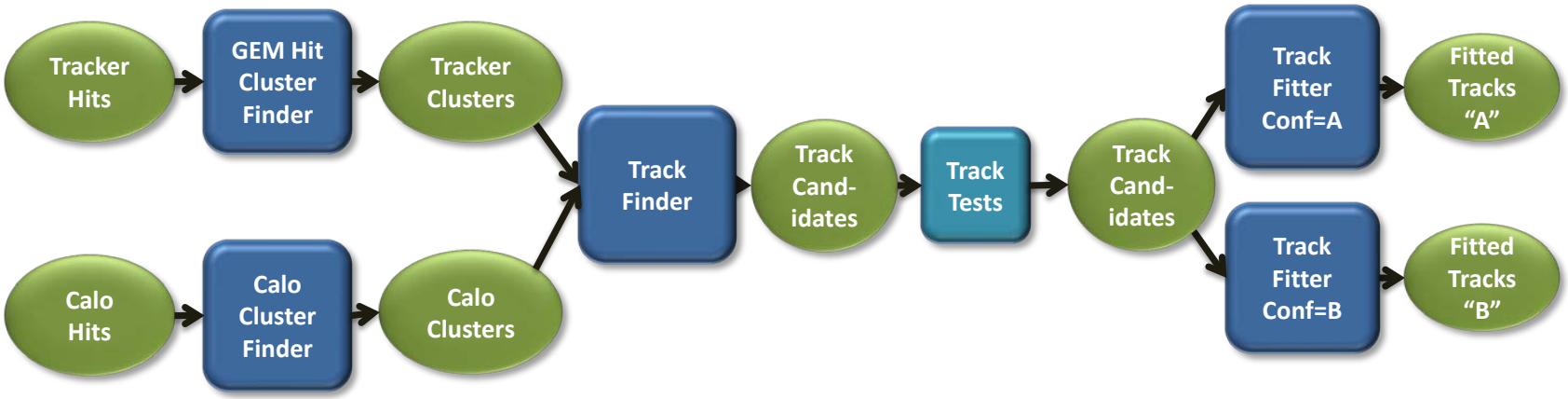
From G. Barrand *et al.*, "GAUDI - A software architecture and framework for building LHCb data processing applications", CHEP2000

Decoupled Algorithms & Data Objects



- Data objects (inputs & results)
  - Mostly "dumb data" (structs)
  - May reference other data objects
  - Hold metadata
- Data consumers/producers (algorithms)
  - Run-time configurable
  - Single algorithm per module

Analysis Chains



- Modules communicate exclusively via data objects
- Module relationships configurable at run time
- Multiple chains per job
- Support for condition testing modules
- Output modules (not shown) for DST and histogram/ntuple files

Frameworks Services Features

Feature	art (FNAL)	CLARA/Hall B	JANA/Hall D
Transient event store	Event, run, sub-run objects	Data “banks”	With producers
Folders in event store	no	no	no
Event Data Service	template function	bank API	template function
Message service	yes	yes?	yes
JobOptions Service	FHiCL API	config messages	ParameterManager
Geant4 integration	artG4	no	no
Detector Data Service (geo)	no (service API)	yes (Java geo model)	JGeometryXML
Detector Data Service (cond)	no (service API)	CCDB	JCalibrationCCDB
Histogram Service	TFileService	yes?	no
Interactive mode	no	no	no
Configuration test	yes	no?	no
Memory tracker	yes	no?	no
Polymorphic data objects	yes	no	yes
Inter-object references	art::Ptr, art::Assns (1-1, 1-N, N-N)	integer indices, link banks	integer indices