

Workflows and Workflow Systems at Fermilab

Jim Kowalkowski, Fermilab Scientific Computing Division

1 Introduction

1.1 Purpose

This paper provides information on the state of the art in distributed area (DA) and *in situ* (IS) workflow management systems from the perspective of the Fermilab Scientific Computing Division (SCD) and the experiments and projects that it is directly affiliated with. The paper summarizes what can and cannot be done today with existing workflow systems and the challenges of the future. It is motivated by the Workflow Workshop panel discussion descriptions, which states

[Provide] a high-level overview or it can be something more specific about some workflow aspect. Some of the questions that one could consider addressing are:

- How can we characterize existing DA and IS workflow systems?
- What are the strengths and weaknesses of the DA and IS workflows?
- What are the common functions in DA and IS workflows?
- What is lacking in current systems?

In addition, we will highlight some future directions.

1.2 Scope

The science tasks of interest here can be divided into three major classes: high energy physics (HEP) experiments, Astrophysics experiments, and specialized simulations. HEP tasks can be broken down into three major classes: detector simulation, production data processing (aka reconstruction), and analysis.

All HEP tasks have traditionally been run on distributed resources and have just recently been moving towards utilizing HPC resources. The work within SCD is dominated by CMS and the upcoming Intensity Frontier experiments that include NOvA, muon g-2, mu2e, and later DUNE. All science tasks from this area will be represented or summarized in this paper.

Astrophysics experiments have been using a mix of distributed and HPC resources, although they are currently dominated by distributed resources. Only analysis work will be represented here from this area. The focus will be DES analysis and some forward-looking analysis work for the LSST Dark Energy Science Collaboration.

The specialized simulations utilize HPC resources. Examples only include Accelerator Modeling with Synergia and LQCD production workflows using MILC.

1.3 Terminology

Event Data Model: Representation of the data that an experiment collects, all derived information, and historical records necessary for reproduction of results.

Event: A collection of data products associated with one time window, the smallest unit of detector data collection to be processed. It can be thought of as all the physical interactions that occurred within a time window throughout an apparatus.

Module: An object that “plugs into” a processing stream and performs a specific task on units of data obtained using the Event Data Model, independent of other running modules.

Data product: Experiment defined objects that represent detector signals, reconstruction and simulation results, physics objects, etc.

Software framework (SWF): Coordinates *event processing* via configurable, pluggable modules (e.g., reconstruction, filtering, and analysis) that add data to and retrieve data from events, supporting a programming model that separates algorithm and data. Paths that events move through are specified using a simple workflow language. Responsibilities include managing metadata and abstracting I/O actions from the scientist developer-user.

Dataset: Events are collected into files with hierarchical structure and metadata. Files are collected and managed by data handling systems and contained within datasets. Datasets are collections of files that serve a particular purpose: all the raw data an experiment collected up to now, all the simulation produced with a particular physics signal, all the reduced data necessary for producing conference results.

Campaign: All the computing that is necessary to process a dataset or to construct a new dataset or to derive a new dataset from an existing one. Carrying out a campaign is a major goal of a workflow system.

ROOT: The current set of tools used, in this context, for specifying the data model, managing the I/O to files, and providing commonly used physics objects, such as histograms and tabular data containers. It also provides a library of commonly used HEP algorithms for aiding in the analysis and display of data.

Geant4: The HEP standard simulation engine for propagating particles through matter. In this context, it is embedded and managed within the software framework.

2 Overview

2.1 HEP state of the art

A general view of the science problem being solved within an HEP experiment is depicted in Figure 1. Nearly all HEP workflows related to production and data processing follow this form. Production processing will have one or more phases that include simulation and data reduction. Workflows will typically be realized in two major layers: a campaign specification that outlines the overall goal that is to be achieved, and a configuration for a fine-grained workflow engine within the software framework that operates on partitions or blocks of data from the ensemble of data to be processed. The software framework coordinates most of the actual algorithmic work that is carried out. Almost all of the simulation and data reduction sub-workflows are carried out within software framework applications. The format and interpretation of data files up through “reduced data” follow a standardized ROOT format. The framework defines an interface for each algorithmic module to fetch and put data into an event.

An abstract view of a campaign workflow is depicted in Figure 2. A typical workflow specification will describe these steps, connections, and data sources. The overall goal is to start from nothing but a description of the physics that we want to model and produce a number of datasets, some of which are optional: (a) particle data or the output from the generator, (b) instrument response dataset or output from the simulator, (c) derived dataset or the output from running data reduction algorithms, and (d) additional derived datasets that contain only key features for various physics analysis.

Types of HEP Processing Tasks

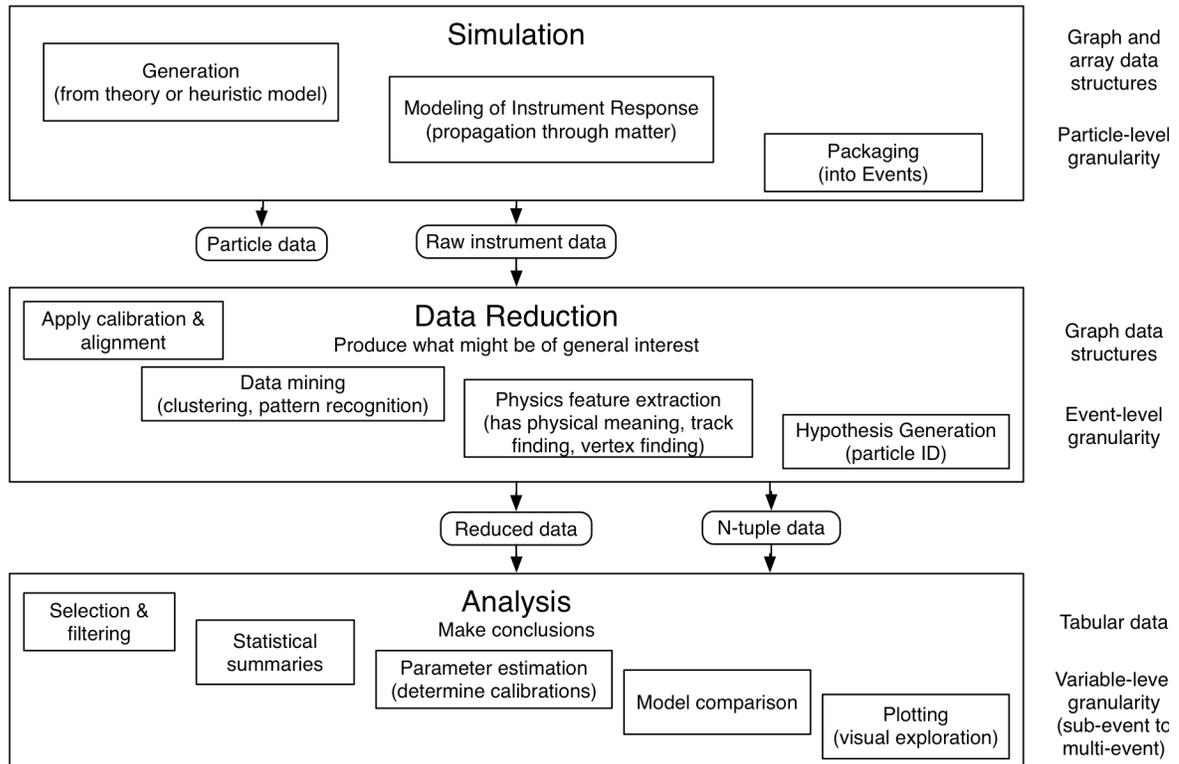


Figure 1 – Types of HEP Processing Tasks

As with many production processing systems, there are several important features that make specifying a workflow at this level difficult. Each of the phases could require hundreds of thousands of executions of the software framework to carry out the configured task. In the simplest form that is still common, each phase has a boundary that requires all files contributing to a dataset to be synchronized with the global permanent storage and data handling system. Input data files must be properly staged before the computation is scheduled to avoid wasting compute cycles waiting for data. The auxiliary data access requires connections to high performance databases, hierarchical caching system, or staging technology depending on the size and complexity of the this kind of data.

Resource needs and mappings for the campaign are not specified at this level. Another key feature is that the actual physics workflow configuration is buried at a lower level. That workflow specification is also used during testing to validate results. It also ensures data object type consistency when chaining algorithms. The metadata protocols provide information flow from the internal framework out to the data handling systems.

A typical expanded view of any of the processing boxes can be seen in Figure 3. Here we see workflow jobs scheduled where the data lives or where standard data movement tools have been used to migrate data into place for processing. The generic software framework is acting in two roles: running the algorithms (CPU bound) and merging results (I/O bound). The SWF step traditionally consumes one batch “slot” or core during its lifetime, which is

Abstract production workflow

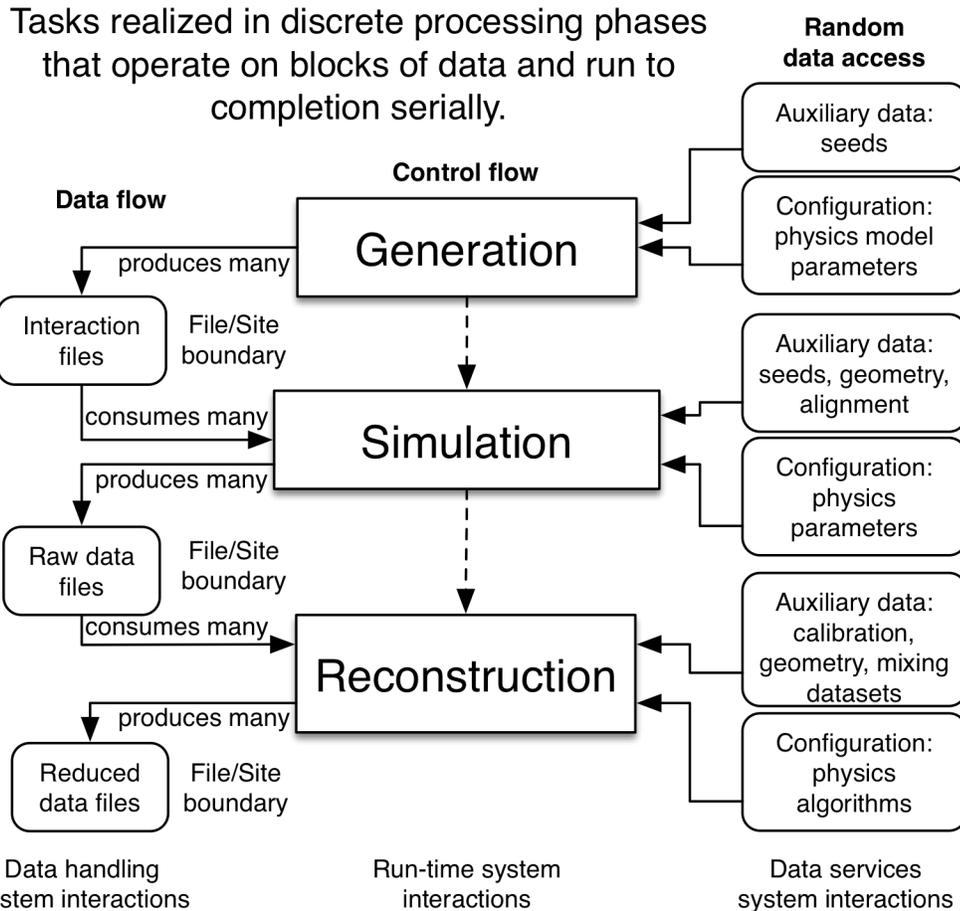


Figure 2 – Abstract production workflow

aligned with the time a resource can be reasonably held (about 8 hours). Output data is transferred to a safe location in units of files. Note that a merge step cannot start until enough data files appear from many other SWF jobs. This makes specification with job DAG tools difficult.

2.1.1 Simplified view of the CMS layers

Now we are ready to show a typical workflow specification and the layers of tools and software that are involved in breaking down the campaign into chunks of work that can be completed within a job. CMS is used as an example here in Figure 4 and will be compared with other solutions in a later section. Given a description of the physics necessary to configure the software framework, a dataset to be processed, and software release

information, one can produce a description, or workflow specification of the desired phases and output datasets that need to be coordinated.

Expanding a phase

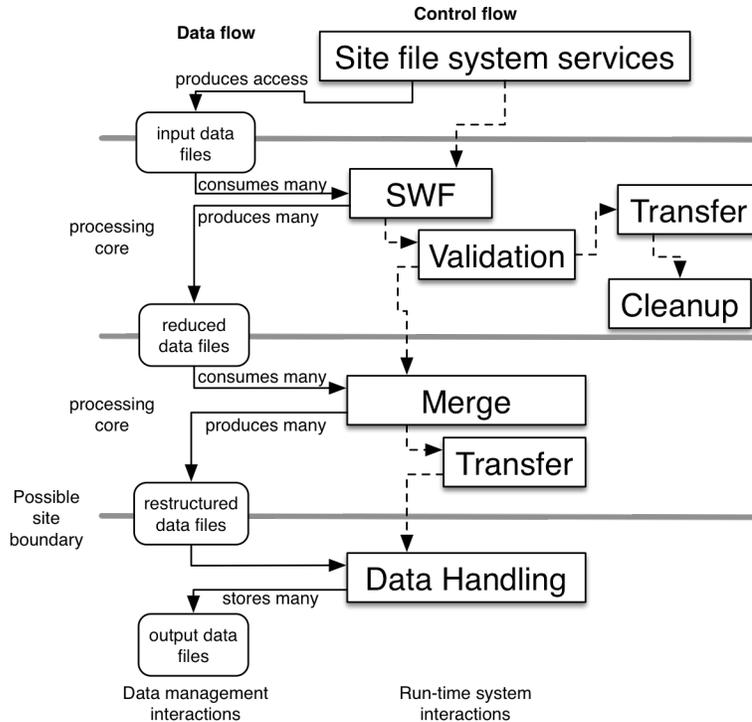


Figure 3 - Expanding a phase

will see good agreement in the overall workflow structure and in the underlying tools that aid in the execution. For underlying infrastructure, this includes job submission, data transfer, data cataloging, data storage, batch systems, and pilot-based workload management. The main differences come about in the orchestration layers of the workflow system. As with the CMS example, each system is tied to the experiment software, its practices and size, and specific physics needs within the software framework. Nearly all use general-purpose scripting languages to directly coordinate activities and communicate with outside systems.

Smaller experiments are moving towards handling the orchestration through global state machines where transitions are triggered by the posting of files into the managed datasets.

2.1.3 Common variations and hidden conditions

Both within and among the experiments there are common variations to the generic workflow given above. Relevant ones for this discussion are: multi-stage generation or simulation, mixing, error handling, and validation handling.

Event mixing happens when noise or other addition signals such as pile-up need to be added to events to make them look more like real data that would be seen in a detector. The typical mixing procedure is to merge many noise events from alternative datasets into the events that are processed in the main data stream.

Much of Fermilab processing is managed at a low level using pilot-based glideinWMS to efficiently handle the allocation and supplying of work to batch system resources managed by Condor. Multiple queueing layers break down work into packages which operate on dataset blocks and break packages into jobs which operate on a few files. Work managers generate jobs that allow maximum utilization of the resource allocations and fair use across competing campaigns and individual analysis tasks.

2.1.2 Differences between experiments

Going from CMS to muon (mu2e, muon g-2) to neutrino experiments (NOvA, MicroBooNE, DUNE), you

Simplified CMS “integrated solution” workflow system

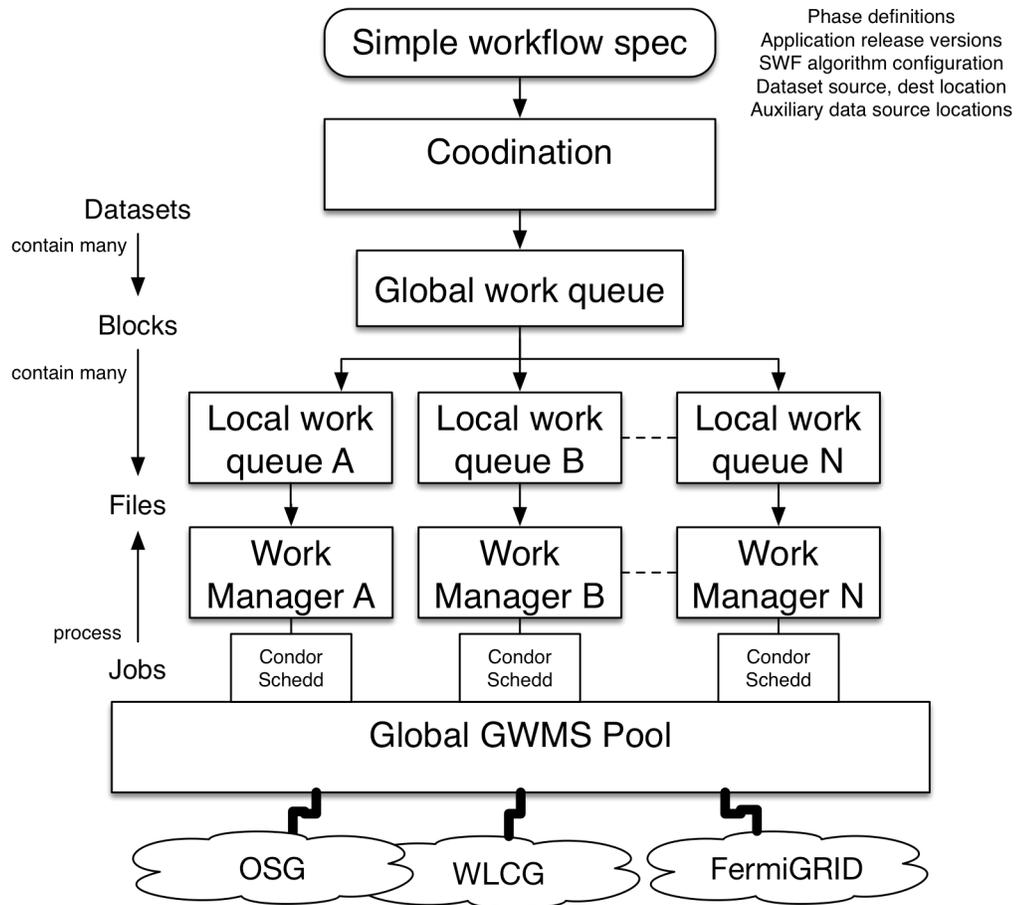


Figure 4 - Simplified CMS workflow system

Any of the major phases may be further broken down into multiple steps. Examples that can cause this are use of tools that generate data files outside the standard formats, or complexity in the mixing process.

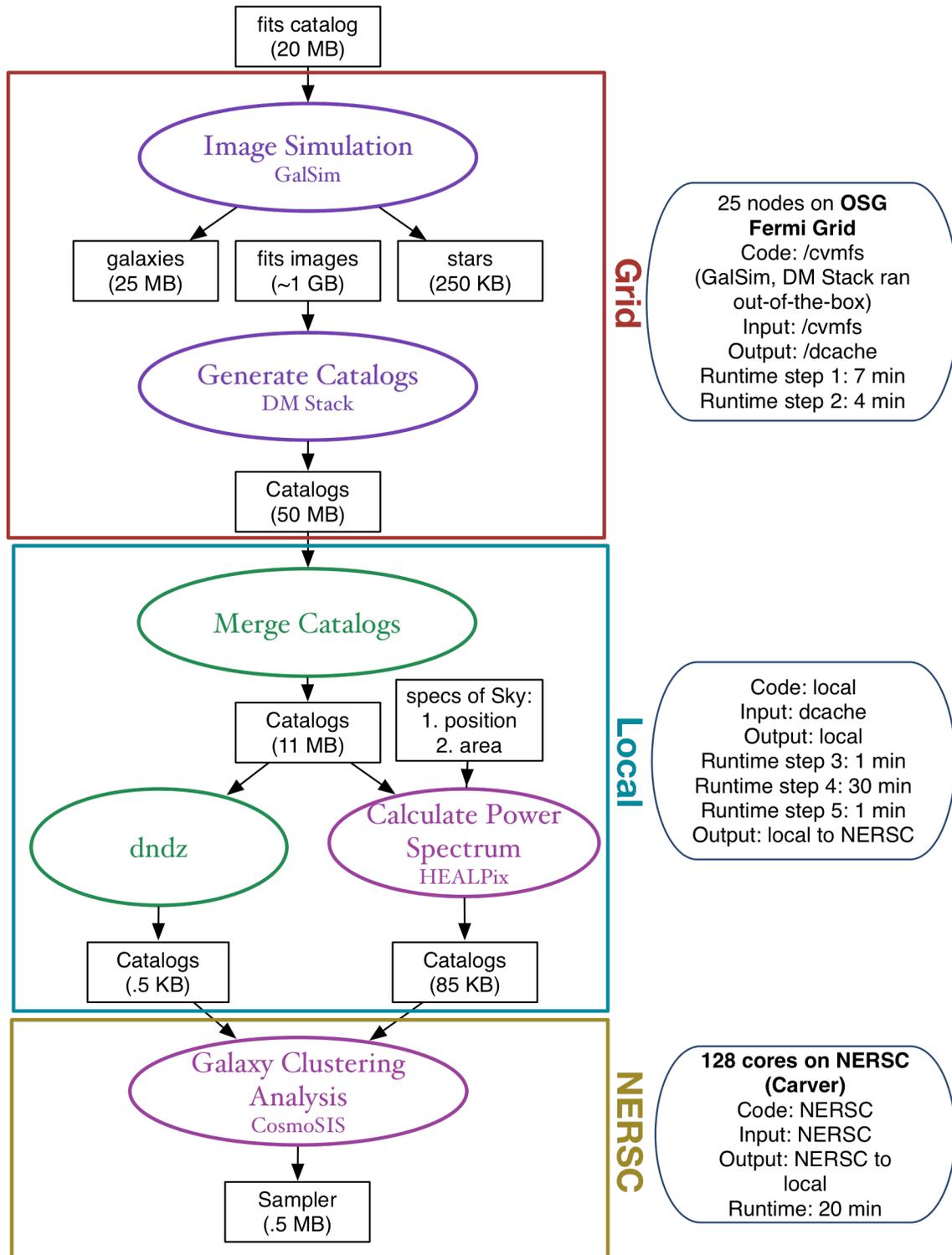


Figure 5 - A cosmology analysis workflow

Some workflows will include paths that run through all phases with a subset of the data that is being generated to verify that the results are correct. Downstream steps are started as soon as enough data appears for it to start.

2.2 LSST DESC

Many years ago, SCD software experts partnered with scientists and developers in the cosmic frontier (astrophysics area). One of the major goals was to bring the collider / detector collaborative software best practices and tools into the cosmic area to help address software development needs within their rapidly growing collaborations. Early efforts start with the now-deceased JDEM project with the Science Operation Center. More recently the team worked with ANL on the PDACS projects, and then on the CosmoSIS project with DES. The latest project was with the LSST Dark Energy Science Collaboration demonstrating analysis workflows that include CosmoSIS and carrying forward what was learned in PDACS and the JDEM SOC. This demonstration, shown in Figure 5, is interesting here because it has elements of the standard HEP workflow deployed using multiple computation sites, using common FNAL-supported batch processing infrastructure, community-developed science applications, and a general portal system with workflow capabilities (Galaxy carried forward from PDACS experience). In addition, it includes running a generic software framework for parameter estimation (CosmoSIS). The analysis of this project is outside the scope of this paper. It did, however, provide a good test case for exercising a generic workflow management system with a real science analysis use case involving a diverse set of applications. The demonstration showed promising results for canned or commonly used workflows, where well-defined parameters are exposed through the application wrapper layer and controlled through a web interface. The most interesting result came out of postmortem discussions. The scientists involved wanted to see much closer integration with the development and run-time environments, where repositories are manipulated, code is changed, and local testing is accomplished. This includes workflow specifications written in text files and quick integration with other applications contributed by collaborators. This very much matches the requirements and work modes of the HEP detector/collider community.

3 Commentary

Here I only give a few examples of changes, concerns, and challenges heading into the future for workflow systems that directly affect the Fermilab program.

3.1.1 Near future changes

File transfers to global storage are being avoided by scheduling workflow stages back-to-back on the same resource. Files are written to and read from local storage during these stages and only moved to permanent storage at well-chosen end points. There are obvious benefits for this organization, but there are also tradeoffs in the amount of data that can be processed due to application run-time limits and longer chains mean longer run-times.

With multicore capabilities entering the software frameworks, as with CMS in the run that has just started, entire nodes will be allocated for this multi-stage processing to help optimize and balance performance.

WAN Networking infrastructure improvements have permitted streaming capabilities to be integrated directly into the front-end of the software framework applications using technology such as XROOTD.

3.1.2 Future challenges

Over the next ten years, the bigger challenge will be to process data coming from increasing complex detector systems. CMS will be realizing a pile-up increases of 5-7 (pile-up of 140-200). Increases in data rate and complexity can require increases in CPU resources

as much as 50x without substantial change considering the $O(n^2)$ tracking algorithms. At the Intensity Frontier, extremely large LAr detectors will experience similar tracking algorithm difficulties.

Analysis follows standard production runs. Working groups and individuals perform these activities. Current practice is for these activities to be coordinated through batch systems, using resources that overlap with production processing. An obvious goal is to move these processing times down towards interactive system speed.

3.2 Unique orchestration solutions

Every experiment automates standard or common workflows for the production processing described above and for some of the later analysis workflows that follow. This automation comes about through experiment-provided scripting and experiment-developed tools. Sharing or commonality seems to come mainly at the level of “the job”, and the tools necessary to make the software and data available at the computation site. This is true across almost all the classes of processing seen at Fermilab. Why has common tool practices and sharing using external tools not extended to any reasonable degree into the orchestration layers? Here is a list of some of the major things that collectively give insight into the answer of this question.

- Workflow specification language capabilities – The scientists here want text files that can be modified with the editor of their choice. XML is too verbose. Graphical editors are not liked. A custom language needs to express the things they want clearly: data-flow as well as control flow expressions, good user-defined type support, simple and algorithm configuration constructs. Type handling and configuration have in the past not been able to satisfy the needs of an experiment better than what specialized scripting can do. The abstraction level is an important feature and how it maps onto underlying systems. This is one of the more difficult areas to gather good requirements.
- Application integration – the languages, protocols, and constructs necessary for binding the science applications to the workflow system have not been well-received or readily done by developers that are not experts in the workflow system tools.
- Mapping and integrating additions features input workflow systems, such as monitoring integration, automated provenance collection and propagation, and auxiliary data service interactions.
- Scaling down for use in a development environment on a laptop and cleanly scaling up to local clusters and GRID resources without substantial infrastructure resources and administration. Control over translating workflow specifications to resources.
- The existence of flexible software frameworks that provide the applications.

3.3 Strengths and weaknesses

One of the key strengths that have made large-scale science possible is the collaborative development systems and the establishment of protocols within the software framework applications for algorithms interacting with data and the rest of the world. In some respects, it is due to this common programming model and architecture. This model has permitted hundreds of collaborators to contribute algorithms (as pluggable modules) into common libraries and repositories. These algorithms are dynamically combined into comprehensible processing sequences at run-time using simple scripting language commands. The dynamic configuration of sequences can contain thousands of modules that carry out complex science tasks within one program execution. It is worth noting that nearly all the algorithms are heavily parameterized (using a standard configuration language) to accommodate con-

ditions that will be experienced during operations. All of these features do mean that there are relationships and integration of development, packaging and runtime environments.

It is my observation that there is a fairly large barrier between the internal software framework orchestration and the higher-level campaign orchestration layer. There is almost no visibility into the scheduling choices between these layers. This “decoupling” limits opportunities for reordering, splitting, or joining workflow steps to better utilize the available computing resources. Up until now, this has been a reasonable limitation with the assumption of higher uniform processing nodes. Software frameworks also produce important performance statistics, status message, and provenance records that need to be communicated and interpreted at all levels of processing. This information helps to identify errors and bottlenecks and can help to steer future workflow steps (based on announcements of data file completions). The decoupling locks formats down at the “log file” level, where this out-of-and data must be moved off the processing node and parsed. A more integrated approach could provide commonality in the protocols and data formats at the transaction level, therefore allowing more flexibility for use. An advantage to this organization is that “scaling down” to the laptop or development system level for creating modules, testing, debugging, and personal analysis is very natural.

Because orchestration layers are either hand-written or produced specifically for an experiment, the operations that can be performed, the applications that can be used, and the underlying systems that can be used have a rigid specification. In other words, specific applications must be changes to reflect different high-level workflows. This is generally a bad quality, but it does mean that the system does what the experiment wants. In my opinion, another disadvantage of this organization is that the system are difficult to evolve if there are changes in computing technology because the major development is complete when the experiment begins to take data. Upgrades to existing experiments do allow for some adaptation.

Metering of work due to current computing models means long term-around time.

Complexity in the I/O and persistency layers within the framework and the costs of file-oriented processing has the advantage of allowing for user-level arbitrarily complex data models that do not impact the framework software. This is great for tailoring data structures to match algorithms. It also permits reuse of the technology to store provenance and other metadata within the file alongside the main data stream. The automation here does come with a cost. Arbitrary data structures can be difficult to pass from machine to machine without serialization and utilizing the file protocol. Performance is also difficult to predict.

There are other features of the module protocol and event data interactions that make scaling and scheduling of calculations difficult that are outside the scope of this paper.

3.4 R&D Challenges

As stated earlier, WAN networking improvements have definitely helped make large-scale workflow processing possible. The improvements on the standard heavy-weight core cluster node have not been that significant with regards to memory, local I/O, and CPU / multicore performance. Movement towards HPC resources has already started and many of the processing tasks outlined here could greatly benefit from efficient access to exascale resources.

One key challenge will be to evolve the programming model to accommodate more diverse processing environments, with special focus on the availability of high performance interconnects and many-core compute resources and heterogeneous architectures. This needs

to be done while preserving the features essential for large-scale collaborative science, which include dynamic configuration that integrates well with the development environment. Currently practice utilizes Object Oriented programming principals to bind parameters to algorithms and internal services to provide random access to auxiliary data. Including a level of parameterization that accounts for parallel resources to be utilized changes this model substantially.

Moving to exascale era machines will almost certainly mean changing from a file-oriented I/O subsystem to an streaming-based system, avoiding expensive object serialization, compression, and permanent storage operations – especially to global off-site storage until they are needed. At some point it may become less expensive to regenerate intermediate data from provenance and metadata stored with derived results, than to request data transfers from permanent storage systems. Moving to efficient streaming-based I/O may also force data structures to be simplified.

Figure 6 shows a future framework where file I/O is minimized by directly moving or operating on events or blocks of events. The framework is still in charge of putting together a fully configured set of modules and scheduling the movement of data through them. It will continue to automatically generate and transmit provenance and metadata so that any relevant piece can be reproduced. These modifications to workflow will require rethinking the boundaries between traditional workflow system components *i.e.* the workload management and software framework, where these components become more tightly integrated to permit far better optimizations.

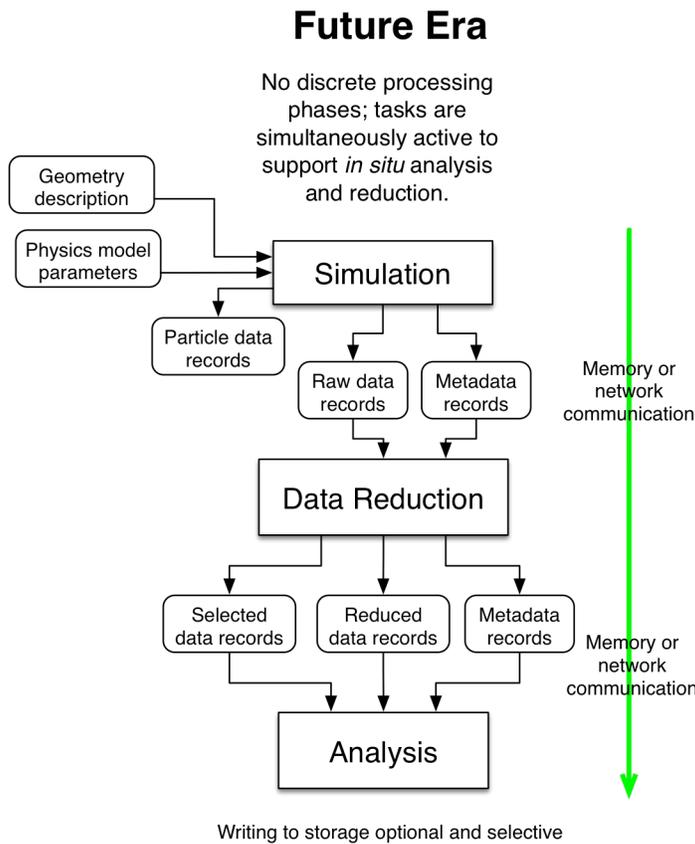


Figure 6 - Future era

4 Conclusion

The “state of the art” workflow management systems of HEP are moving further away from the standard batch cluster node operations due to advances in multi-core, many-core, and networking advancements. Streaming I/O layers and new programming models that allow for increased flexibility in the units of work being processed make these kinds of applications more ready to effectively use exascale era computing resources.

Traditional HPC applications also seem to adding features such as *in situ* analysis processing to permit effective operation in the new era. If I look back at the analysis box of Figure 1, there might be significant overlap in the kinds of functions that scientists will want to do *in situ* and those from the HEP DA world. If this

is true, it could mean that the HPC applications will start facing similar difficulties with irregular data structures and workloads that analysis activities typically involve.

Many of the features necessary for *in situ* processing are readily visible in the online data acquisition system for an experiment. For modern experiments, the software frameworks sit in the real-time data streams and provide the fast reconstruction, analysis, and filter stages necessary to reduce data written to manageable levels.