Automation and Quality Assurance of the Production Cycle

L Hajdu, L Didenko and J Lauret

Physics Department, Brookhaven National Laboratory, Upton, NY 11973-5000, USA

E-mail: lbhajdu@bnl.gov

Abstract. Processing datasets on the order of tens of terabytes is an onerous task, faced by production coordinators everywhere. Users solicit data productions and, especially for simulation data, the vast amount of parameters (and sometime incomplete requests) point at the need for a tracking, control and archiving all requests made so a coordinated handling could be made by the production team. With the advent of grid computing the parallel processing power has increased but traceability has also become increasing problematic due to the heterogeneous nature of Grids. Any one of a number of components may fail invalidating the job or execution flow in various stages of completion and re-submission of a few of the multitude of jobs (keeping the entire dataset production consistency) a difficult and tedious process. From the definition of the workflow to its execution, there is a strong need for validation, tracking, monitoring and reporting of problems. To ease the process of requesting production workflow, STAR has implemented several components addressing the full workflow consistency. A Web based online submission request module, implemented using Drupal's Content Management System API, enforces ahead that all parameters are described in advance in a uniform fashion. Upon submission, all jobs are independently tracked and (sometime experiment-specific) discrepancies are detected and recorded providing detailed information on where/how/when the job failed. Aggregate information on success and failure are also provided in near real-time.

1. Introduction

In High Energy and Nuclear Physics (HENP) experiments, the process of producing datasets follows many steps, each of which need to be tracked and documented to ensure reproducibility and accuracy of physics results. The STAR experiment (Solenoidal Tracker at the Relativistic Heavy Ion Collider) [1] has a well-defined and increasingly automated procedure to handle the generation of Monte Carlo datasets. The steps used in the simulation production process of figure 1 and STAR's tools for automating the process are described below.

In STAR, a dataset starts its existence as a request from a physics working group, which parameterizes the physics they wish to study and the amount of data they will need in order to derive results. These parameters are then approved by the physics working group convener (PWGC) who is responsible for passing the request to the production coordinator. The production coordinator, in collaboration with appropriate activity leaders (simulation or computing project leaders), tries to merge similar requests, if feasible, by recommending common sets of parameters for the event generation or the data reconstruction. When the request is finalized, the production coordinator "contextualizes" the request. This involves selecting programs and program arguments and generating scripts and job descriptions to produce the desired data. Most experiments have at least primitive scripts with iterative methods to produce the required jobs from these requests and submit them to a batch or resource management system, either Grid-based or local. After the output is recovered from the batch system, it must be validated. Any detected failures will fall into one of two categories external failures and internal failures. Causes of external failures can include defective batch nodes, random crashes, network failures and so on. Jobs suffering run-time failures but which have passed code quality pre-screening via regression test can typically be resubmitted without any modification to the failed jobs. Some of the job configuration errors, in contrast, will lead to repeated failures if the underlying configuration error is not corrected: causes can include scripting errors, and improper physics parameters that lead to job crashes or meaningless results. The worst case scenario are human errors in the parameters which would allow the jobs to run but produce un-usable results. Hence, documenting and reproducibility are fundamental to our approach: before and after the dataset is produced, the steps taken to produce them must be documented and the resulting files must be cataloged. Completing the cycle, the physics working group is then informed that the request is finished. Following through on multiple simultaneous requests can demand a lot of time from the production coordinator and is prone to human error in many places. To reduce the burden and increase the success rate of the production cycle, STAR has developed and implemented a suite of integrated software tools for use throughout this process.

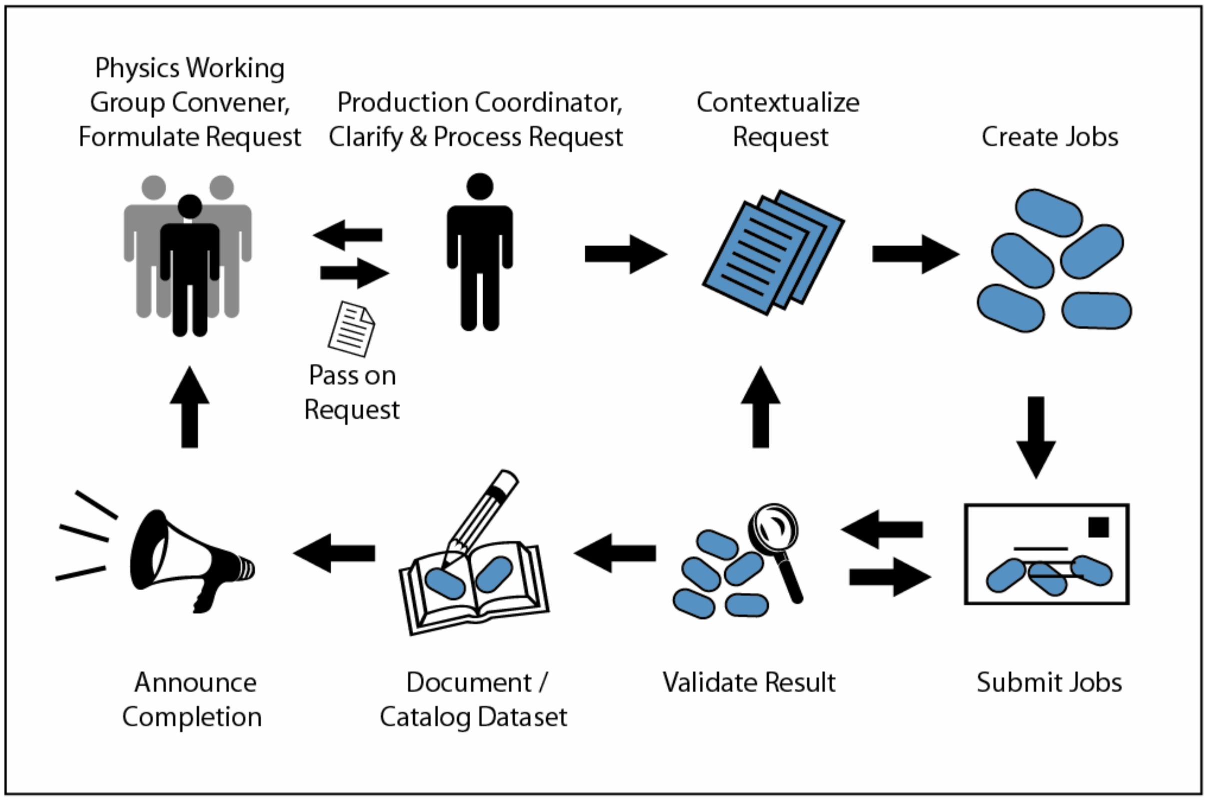
Physics Working Group Convener, Formulate Request
Production Coordinator, Clarify & Process Request
Contextualize Request
Create Jobs
Pass on Request
Announce Completion
Document / Catalog Dataset
Validate Result
Submit Jobs

Figure 1. The STAR production's life cycle and worflow.

2. Request description

There are many parameters to be specified for the jobs to be produced. Without some level of automation much tedious back and forth communication may be required between the production coordinator and the physics working group coordinator with no good away to track amendments to the request. To improve the efficiency of the request process, a simulation/production request API was developed using the Drupal [2] content management system as its framework.. Drupal is the main content management system for STAR's collaborative Web content. It is written in PHP, which is a common language for web development, so developers are readily available. Only physics working group conveners are allowed to request productions, which is enforced by Drupal's ACL structure. As the core Drupal code is well-designed, cyber security concerns are minimal. The web interface GUI provides guidance to the requestors, while using logic to perform input validation, such as making sure required fields are filled in and properly formatted. Users' requests are written to a database and are subsequently easily viewable online, easing the documentation burden for the production coordinator and standardizing the documentation format. The web interface has comment fields allowing the user to request unanticipated options. These fields are only human parse-able, so the human element is still present if needed of course. The production coordinator uses this data to refine and derive the request and converge toward a set of jobs to be submitted. Once submitted, the production coordinator is the only one who can change the request, ensuring that no changes occur in mid-production.

3. Contextualizing the request

The request is transformed by the production coordinator into a JDL (Job Description Language) request which is understood by the STAR Unified Meta Scheduler (SUMS) [3]. This is a short XML file that describes the jobs to be produced. Depending on the type of work the production coordinator who is doing the XML can describe a number of identical jobs with different seeds, or something more complex such as a type of job that encapsulates a subset of the dataset to be processed.

To assist the production coordinator, SUMS provides a multitude of environment variables that provide convenient access to items such as unique seeds for random number generators, strings common to all jobs in this request (but unique to any other request ever submitted), relative job IDs within a request and more. Such variables are useful for creating folders storing data from concurrently running jobs without fear of overwriting, running jobs in a highly parallel manner without the risk of using the same event generator seed or accessing the relative job position within a task.

```
<?xml version="1.0" encoding="utf-8" ?>
<job filePerHour="3.2" maxFilesPerProcess="30" minFilesPerProcess="10">
       <command>
              command1 -agrs
              command2 -agrs $JOBINDEX $FILELIST
       </command>
       <stdout URL="file:./$JOBID/$JOBINDEX.out" />
       <stderr URL="file:./$JOBID/$JOBINDEX.err" />
       <input  URL="filelist:./MyDataset.txt"   />
       <sandBox>
              <package>
                     <file>file:./myfiles/*</file>
              </package>
       </sandBox>
       <output fromScratch="*.out" toURL="$SUBMITTINGDIRECTORY"/>
</job>
```

Figure 2. A JDL to process a dataset with an input file list.

Figure 2 shows a simple SUMS JDL that will create jobs, each processing a few files from MyDataset.txt where MyDataset.txt contains an ASCII line-delimited list of input files. SUMS will reorder this list if requested (this example does not have this option) and split the file list into sub-file lists with each list containing no more then *maxFilesPerProcess* and no less then `minFilesPerProcess`. The number of splits of the file list determines the number of jobs that will be produced. Each job will have a pointer to its unique file list.

The file descriptors can be Physical File Names (PFN) or Logical File Names (LFN). SUMS can understand certain file naming schemes and has a mode that will group all files on the same node together and request that the batch system send all jobs to the nodes on which these files reside (for sites with a simple distributed storage model). Even in an xrootd [4] system, SUMS can declare a soft preference for a particular node to possibly conserve network bandwidth. Such optimizations are typically much too tedious to be attempted manually, but are natural for SUMS to apply transparently.

The SUMS JDL is the same format for independent user analysis at most STAR sites, such as NERSC's PDSF, Wayne State University and MIT. SUMS is deployed in the standard STAR software stack and interfaces with most common batch systems such as LSF, SGE, Condor, X-Grid [5] and the Condor-G grid interface.

4. Job feeder

When the request is provided to SUMS there is an option to dispatch jobs immediately or to create all necessary files without dispatching them. In both cases a session file is produced that describes each job so any job can be generated and dispatched or (re-dispatched if needed) at any time in the future. Large production runs will use the option not to dispatch all jobs immediately. From this point a specialized tool called "the feeder" is used to submit jobs. The feeder places limits on the number of running and idle jobs in the batch processing system at any one time. Once it has reached its limits it stops submitting and marks inside the session file all jobs that have already been submitted and waits until the queue size is reduced before submitting more. The flowchart of the behavior of the feeder can be seen in figure 3.

.

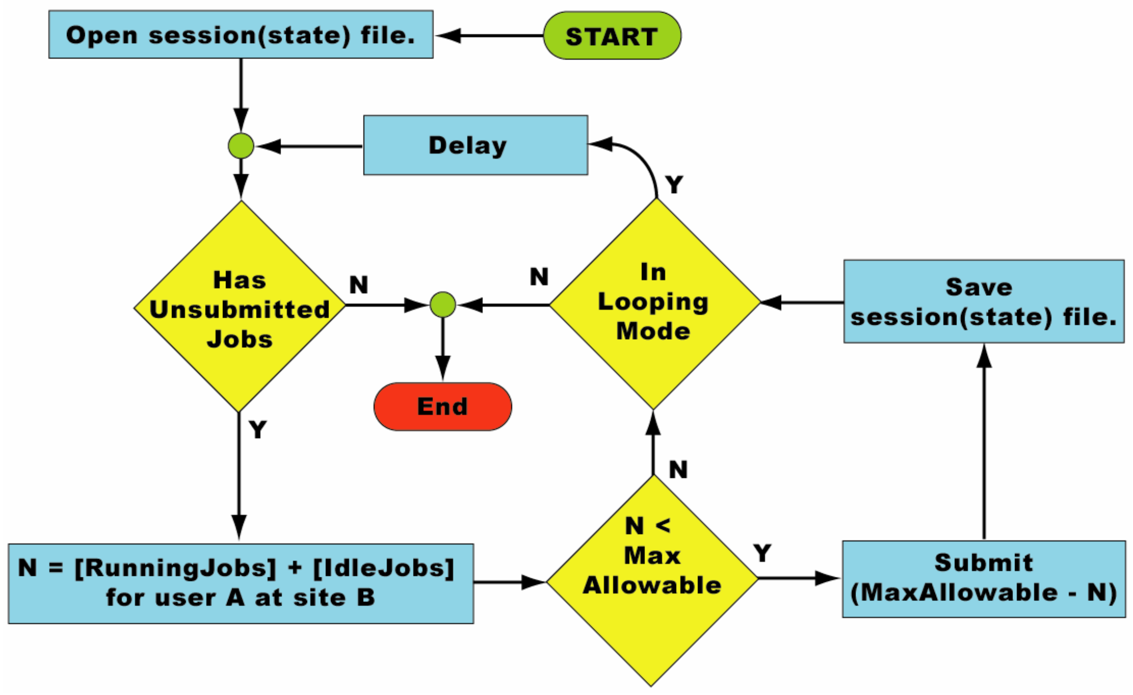
START
Open session(state) file.
Delay
Y
Has Unsubmitted Jobs
N
N
In Looping Mode
Save session(state) file.
End
Y
N
N < Max Allowable
Y
N = [RunningJobs] + [IdleJobs] for user A at site B
Submit (MaxAllowable - N)

Figure 3. Flowchart of the job feeder.

The feeder will check the queue periodically. The feeder's function is to keep the queue topped up. It is most commonly configured to keep the queue at max occupancy, but other configurations exist for fine tuning resource usage. This increases scalability for systems that need to keep track of many jobs, and reduces recovery time from failures. The feeder can be stopped or started when the production system is in a state that may cause jobs to be lost. The feeder could automatically detect conditions leading to job losses. Such conditions includes "job black holes" that exist when a resource failure occurs and for example, whenever a file system unexpectedly dismounts causing all jobs to get sucked through the farm quickly without processing their payload or a node having all jobs crash due to lack of memory or unstable system state. In this way the number of jobs exposed to the failure is limited only to the jobs in the batch system at the time the error occurred. It is known, which jobs have already finished and which jobs have not yet been submitted, so the range that has to be examined, to determine where to resume submitting has been isolated, reducing recovery time. If the batch processing system fails or a higher throughput resource becomes available, the feeder allows the operator to change the target of the jobs, redirecting them to the higher throughput resource, without resubmitting or having to check which were the last jobs submitted, because the feeder is keeping track of this in the session file. This feature is particularly useful for lengthy stand-alone productions (simulations, productions not requiring a tight association between computation and storage), because the likelihood of resources failing (and thus requiring a redirect of pending jobs) over a large time period is greater.

5. The job scanner

The feeder simply keeps track of jobs submitted. It cannot determine if the job was successfully processed by the computing node. This responsibility falls to another tool in the toolbox called the job scanner. The scanner's function is to parse through all the output returned from the job, catalog the data in a database and determine if each job succeeded or requires resubmission.

The scanner is pointed to the directory containing the submission files (e.g. condor submit files), which are parsed. Once this is done the scanner knows how many jobs there are in this request, the request ID, the indices of all jobs, the location of the standard output of the executable, the location of the standard error log of the executable, and the executable script itself. These are all opened and parsed. The scanner has a list of regular expression capture groups in its configuration file that it tries to match. Inside the jobs' output files, tags (mostly from echo statements) have been placed to report job progress and diagnose failures. Tags have also been placed to print the location of the output file (once it will be returned to the requesting site) and its MD5 checksums when the file(s) was on the worker node. Once the output of these tags is parsed from the standard output, the location of the output files on the requester's side is known to the scanner. The scanner will attempt to locate the files. They typically cannot be parsed by the scanner because its format is particular to the submitted job, however the file sizes and MD5 checksums will be inspected by the scanner for comparison. All of this information (the echo statement tags, the file MD5 checksums of all the input and output files as they exist(ed) on both the requester's side and the worker nodes' side) are pushed into a database. Pattern recognition is used to find known error states. For example, the error "segmentation violation" in the output of a job which has previously passed all testing in a controlled environment (regression testing before production) is a pattern known to indicate failure with good chance of success in a second resubmission, even without any modification to the job. Such an error pattern can be the result of emulating 32-bit operation on a 64-bit OS while using a single executable in a shared Grid infrastructure, random hardware fault or corrupted input files. The scanner then marks each job as failed or successful in its database and creates a list of all the jobs that have failed. The scanner cannot check if the physics result is meaningful, only that the job was processed and that all files reached their destinations without corruption. This level of quality assurance would be impractical for the production coordinator to perform by hand across thousands of jobs, but the job scanner can run these checks on one thousand jobs in an hour. Cases have even been found where SRM, thought to be one of the most reliable transfer mechanisms, returned corrupted files through factors outside of its control, such as a case when the SRM server crashed due to a lack of memory.

It should be noted that the STAR job scanner has both advantages and disadvantages when compared to other user job monitoring tools. The main difference between the STAR job scanner and the User Centric Job Monitor (UCM) [6] is that the STAR job scanner is not a real-time monitor. Real- time monitors require one or more additional transfer mechanisms to stream information back to a central location accessible from the submitter or error analyzer node. This imposes a real-time requirement, which under heavy load may be subject to delays. Furthermore, analyzing the progress may involve complex logic. The scanner on the contrary uses stream output recovery systems built into the grid and all batch systems and (assuming the job is complete) the treatment of completion or error is solely based on the presence of those output files (hence a simple logic). Real-time monitors have other advantages though, such as being able to detect a job "frozen" at initialization because it cannot copy in its input file. The job scanner would never recover the output stream for examination, or would at least have to wait until the job dies in the queue, or until the run time becomes so long the production coordinator is convinced there is something wrong.

The scanner's database holds a large amount of data on each job which can be used to produce statistics such as the success and failure rates (with or without multiple resubmissions of the job) and transfer times from different sites. These may be rendered into static plots or dynamically rendered in online web hosted graphs. Figure 4 is such a plot captured from the production coordinator's web page showing the percentage of jobs succeeding over time through various stages of job completion.

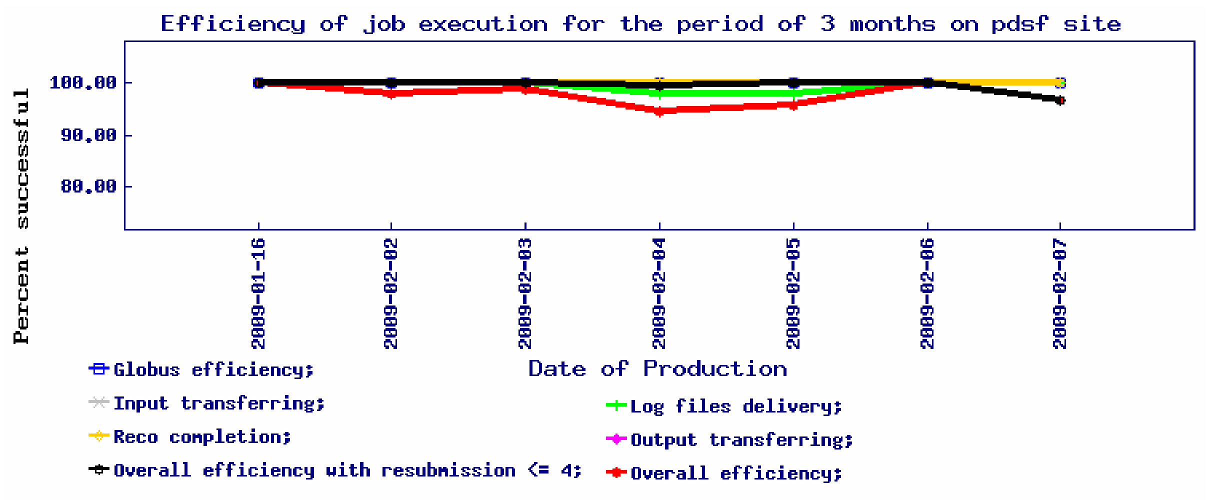

Figure 4. The job scanner's output can be used to automatically generate plots showing the percentage of jobs succeeding through various stages of job progression. The red line is the overall success rate after no more than 4 resubmissions of all jobs.

6. Job resubmission

Once the job scanner has run and produced the list of failed jobs, the production coordinator can examine the cause of the failures and resubmit. With large runs it can be practical to resubmit even without checking the cause of every failure because many will run to completion with a second submission without modification to the job or the environment. Figure 5 shows the total percentage of successfully completed jobs for several datasets subject to multiple passes when needed.

When the scheduler (SUMS) produced the jobs, it created the session file containing all information needed to regenerate and resubmit any job even if the original job description and submission scripts were destroyed. If this file is passed back to SUMS, SUMS can resubmit a subset of the jobs. The job scanner prints out the exact command with arguments required to resubmit the jobs. The reason for not implementing a totally automated resubmission mechanism is because other actions may need to be taken to allow the jobs to run to completion, such as remounting a drive, or rebooting a gatekeeper. Without allowing for this action the jobs could circulate indefinitely being resubmitted after each failure.

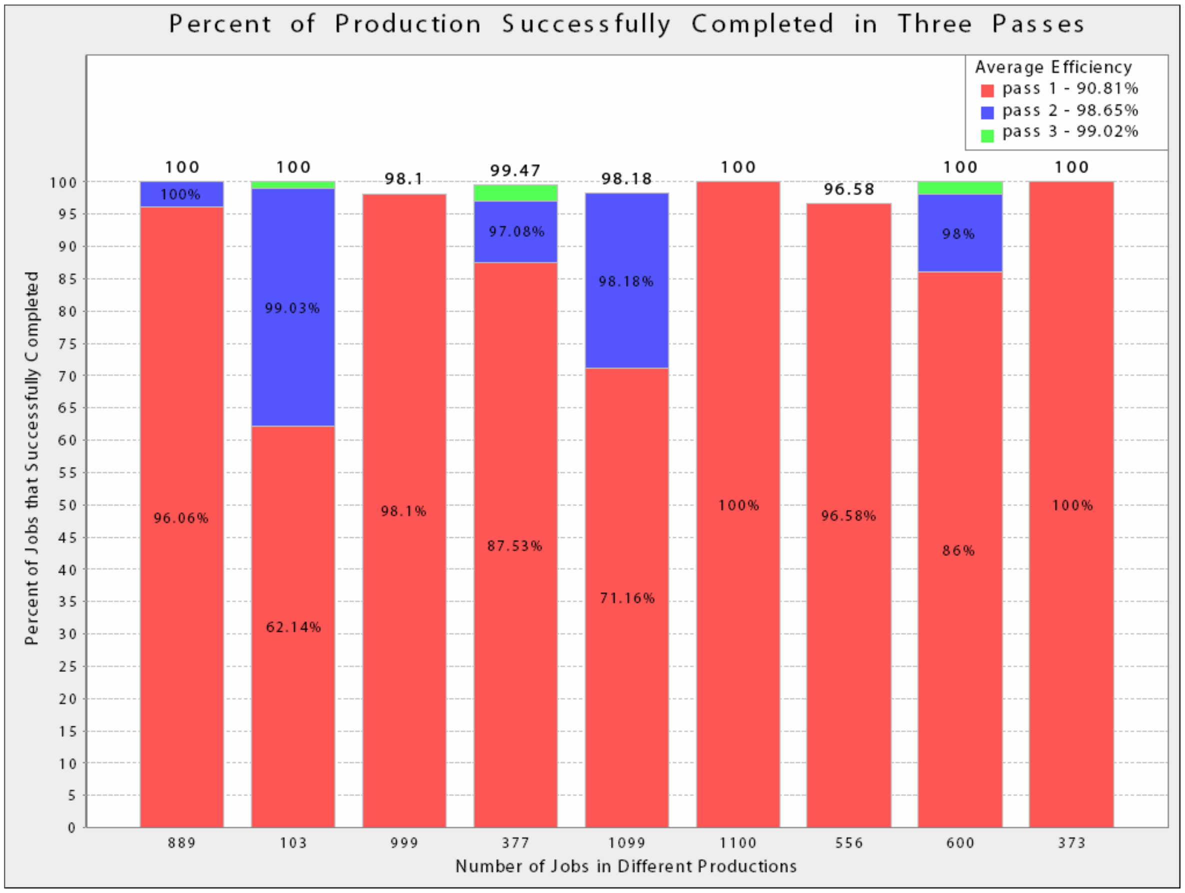

Figure 5. The percent of different productions successfully produced. In each pass the jobs in the dataset where scanned and those jobs the scanner marked as bad where resubmitted, if time permitted.

7. Future directions

The Drupal based production request API is the newest tool in this suite developed to help automation of the production system. It is also the component which has seen the least service. As different types of simulation requests are submitted by the users, fields currently being entered in the comment boxes may be given dedicated fields in the submission form to standardize them. To improve the submission validity checks, more complex logic will be introduced as users provide test cases that may not be adequately handled. The transformation from physics working group request to XML request will remain a manual process, though the software should produce as much of a customized template as possible for a given request type. (In much the same way a JAVA IDE might produce a class file with an empty constructor method if you asked for a new JAVA class).

There has been a steady stream of changes and enhancements to SUMS since its inception over seven years, as with any actively used and supported software. There are no upgrades slated that could improve automation, however many changes are related to expansion of the request description language to allow for the description of ever more complex jobs and work flows. The resubmission features of the scheduler are considered stable and are not expected to change in the near future.

Future changes to the scanner software include identifying less common error states. Once the fault is identified, a regular expression can be added to the configuration file to match the pattern of the error. An intermediate failure recovery mode is also desirable. This would change the jobs' workflow so that jobs would buffer the output on the host site. Thus if the transfer back fails even though the job has otherwise executed successfully, the transfer step alone may be retried, instead of running the entire job from scratch, thereby reducing the load on computing resources.

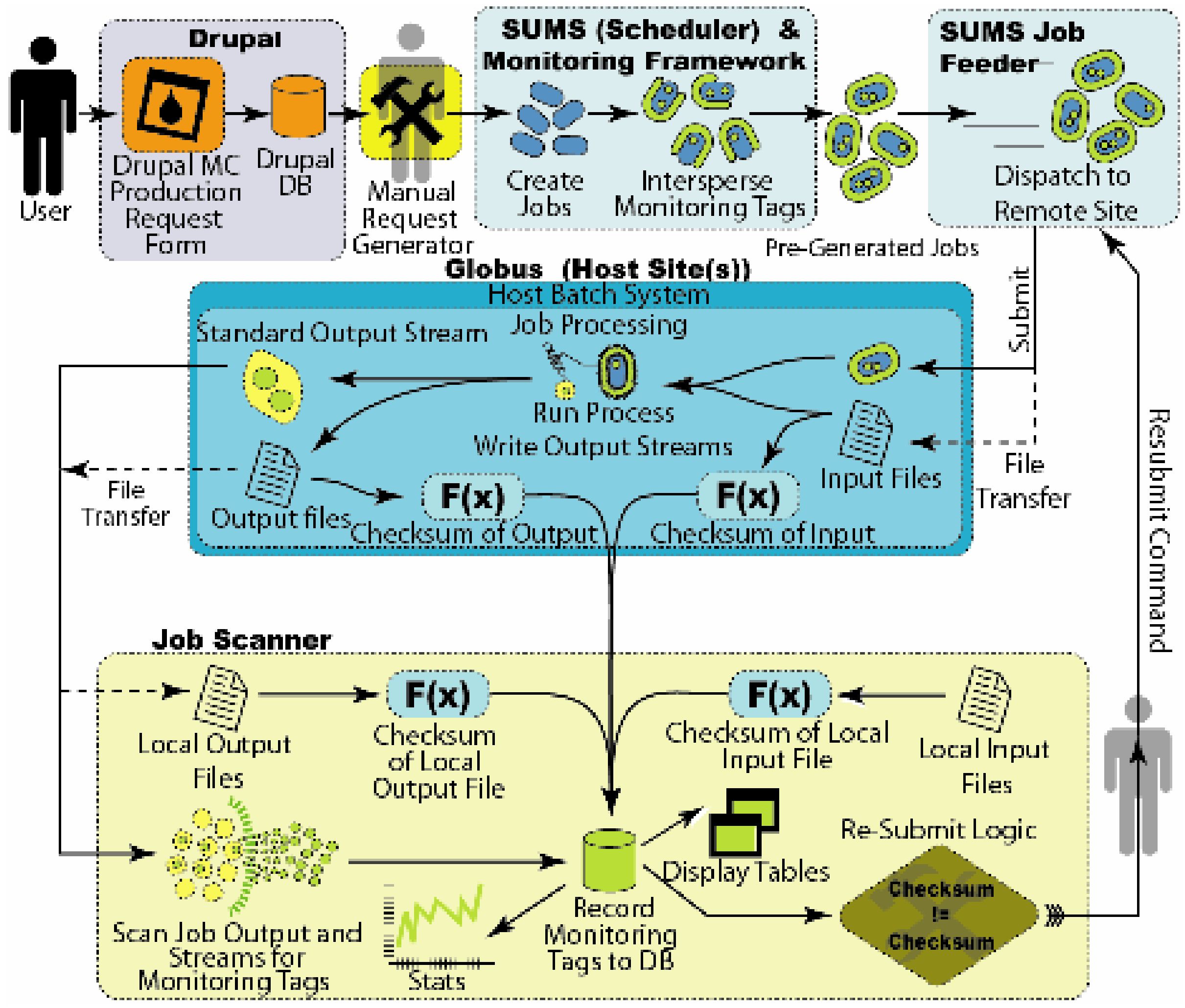

Figure 6. Detailed overview of STAR's simulation production tools.

An overview of how STAR's simulation production tools fit together can be seen in figure 6. STAR finds these tools useful and will continue to devote resources to improve them and share our work and findings with the community if there is interest.

8. Summary

In the task of producing large simulated datasets time is always against the experimental production manager. Reproducibility of results is also a key element essential to a good Physics throughput, comparative research and back-tracing of problems if any. To utilize the available resources to their utmost and assist the production coordinator in producing large simulated datasets, STAR has developed a toolkit aiming to ease the daily work and simply the lifetime and cycle of production. It consists of the Drupal API request submission interface, the SUMS scheduler with resubmission, the job feeder, and the job scanner. The components cover for the requirements of traceability, convenience, automation and reproducibility while preserving roles and responsibilities throughout the production cycle. Especially, the Drupal API interface allows the physics working group to pass a concise validated and self documenting request to the production coordinator. The production coordinator contextualizes that request into a JDL request, which is passed to SUMS to produce jobs and manage them. The jobs are then passed to the job feeder which submits as many jobs, from the production, as the batch system can run at one time. Once the jobs have passed through the batch system the job scanner can sort through the output and mark jobs for reprocessing that failed on the first pass. These jobs are resubmitted by the job feeder and the full process is recorded in a back-end database from the request to the result.

STAR has produced a production software suite to achieve this goal and have used the system in routine production on both Cloud and Grid resources. STAR will continue to refine these tools to extract ever higher efficiency from simulation production runs and make the task of production less burdensome.

9. Acknowledgment

This work was supported by the HENP Divisions of the Office of Science of the U.S. DOE.

References

[1] C. Adler et al. (STAR Collaboration), Phys. Rev. Lett. 89, 202301 (2002)
[2] http://drupal.org/
[3] L Hajdu, J. Lauret. Scheduler Configuration For Multi Site Resource Brokering CHEP06, Computing for High Energy Physics, 2006.
[4] A. Hanushevsky, A. Dorigo, P. Elmer, and F. Furano. The next generation ROOT file server CHEP04, Computing for High Energy Physics CERN, 2004.
[5] L. Hajdu, A. Kocoloski, J. Lauret and M Miller 2008 J. Phys.: Conf. Ser. 119 072018
[6] D.A. Alexander, C. Li1, J. Lauret and V. Fine 2008 J. Phys.: Conf. Ser. 119 052001