Automated and Distributed Monte Carlo Generation for GlueX

Thomas Britton

February 2020

1 Abstract

MCwrapper is a set of systems that manages the entire Monte Carlo production workflow for GlueX and provides standards for how that Monte Carlo is produced. MCwrapper was designed to be able to utilize a variety of batch systems in a way that is relatively transparent to the user, thus enabling users to quickly and easily produce valid simulated data at home institutions worldwide. Additionally, MCwrapper supports an autonomous system that takes user's project submissions via a custom web application. The system then atomizes the project into individual jobs, matches these jobs to resources, and monitors the jobs status. The entire system is managed by a database which tracks almost all facets of the systems from user submissions to the individual jobs themselves. Users can interact with their submitted projects online via a dashboard or, in the case of testing failure, can modify their project requests from a link contained in an automated email. Beginning in 2018 the GlueX Collaboration began to utilize the Open Science Grid (OSG) to handle a bulk of simulation tasks; these tasks are currently being performed on the OSG automatically via MCwrapper. This talk will outline the entire system of MCwrapper, its use cases, and the unique challenges facing the system.

2 Genesis

GlueX is an experiment housed in Hall-D, one of Jefferson Laboratory's four experimental halls and is comprised of an international collaboration of 116 members across 27 different institutions. It collects roughly two PetaBytes of data a year at a rate of approximately one GigaByte each second when running. In order to produce some physics results GlueX relies on Monte Carlo simulation. This simulation workflow involves the precise configuration and running of several different programs each of which can be grouped into 4 major steps (Generation, Geant, Smearing, Reconstruction/Analysis).

Like all good projects MCwrapper was born out of necessity by one Postdoctoral researcher; everyone had their own personal script(s) to run the workflow,

students shared second hand scripts that often had missing options of parameters, many of the parameters involved with the workflow had to be mirrored across several different configuration files. All of this lead to a system prone to human error and unable to provide for proper provenance for any data produced. Not wanting to deal with these intricacies more than once, MCwrapper was created.

Ultimately MCwrapper seeks to be a "one-stop-shop" for simulation in GlueX and Hall-D. To accomplish this MCwrapper must be able to complete the entire production chain, provide basic standards of simulation, accommodate special configurations for individual studies, utilize available batch systems, and provide support for new users. Special attention was given to the utilization of available computational resources, going beyond Jefferson Laboratory's local cluster and enabling users to almost seamlessly utilize the batch systems of their home institutions.

The "engine" of MCwrapper is run by a script (gluex_MC.py) which takes user parameters via both a special configuration file (although MCwrapper is agnostic as to the name of this configuration file it is generically referred to, by the users, as the "MC.config file") as well as the command line. This "engine" actually handles the sourcing of needed resources necessary to complete the workflow and configures the underlying programs, handling outputs as specified. A basic graphical representation of this system is given in Figure 1.

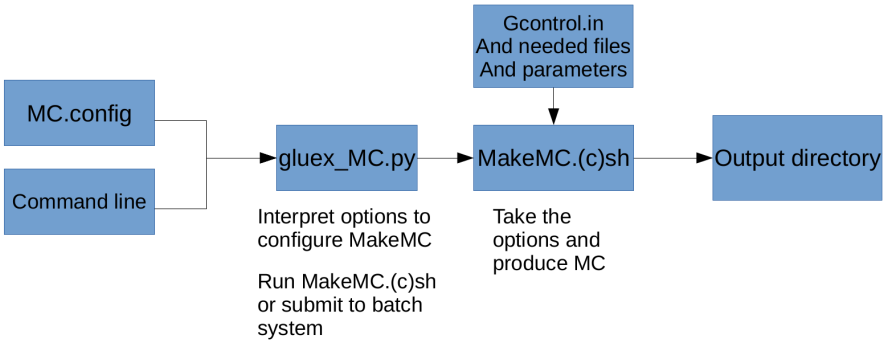

Figure 1: The basic structure of MCwrapper's underlying system. Parameters from MC.config and the command line are fed to gluex_MC which breaks the request into the necessary jobs, configuring one or many instances of MakeMC.(c)sh to run. Each run of MakeMC.(c)sh is responsible for obtaining needed resources, configuring the underlying software packages, and marshalling output.

It is the gluex_MC.py script which contains the knowledge of how to deploy the individual workflows on several underlying batch systems. At the time of CHEP 2019 this list includes PBS, condor, SLURM, as well as a few special instances thereof. These few batch systems cover approximately 90% of collaborator home institutions. There are also two special implementations which can

be utilized by MCwrapper, these implementations cover the Open Science Grid (OSG), which is ultimately based on the condor batch system, and Jefferson Laboratory's own homegrown workflow management system which is based on the SLURM system. Given the workflow knowledge encapsulated by MCwrapper there are minimal changes a user must make to configure MCwrapper to use one or another system (e.g. configuring MCwrapper to run on the OSG versus at Jefferson lab requires changing a single string in the MC.config file).

3 Towards Automation

After the integration of submissions to the OSG there quickly grew a demand for centralized production. Growing tired of managing everyone's simulation by hand on the OSG the flexibility of MCwrapper could be exploited to automatically manage Monte Carlo production. The automatic arm of MCwrapper (referred to as MCwrapper-bot) desired to maintain an extremely low barrier to entry. To accomplish this the flexibility had to be restricted (users have always been able to produce their own simulations with all of the power and flexibility) but in exchange users need answer only a minimal set of questions and gain the benefits automation can offer, including easy access to the OSG as well as automatic job monitoring and re-submission.

Essentially, MCwrapper-bot is just a wrapper around MCwrapper. This single abstraction allows for the creation of a web-based interface to a central production system. The dynamic request submission interface is shown in Figure 2. The added integration with other GlueX systems means the barrier to enter stays incredibly low. A new graduate student can easily put in a request and produce valid MonteCarlo simulations without having a deeper knowledge of individual configurations needed. This is accomplished by specifically building an interface that presents options in plain text (e.g. the analysis submit form accessible from the submit page), dynamically showing or hiding specific options to reduce total form complexity, and localizing all configuration parameters to a single place.

After submission users receive an email confirming receipt of the request and are presented with a link to a dashboard (Figure 3 that shows system statistics, projects which progress dynamically, and gives users, and administrators, the ability to interact with active projects. Projects then automatically test run a small sample locally with the same software stack as requested to be used to produce the Monte Carlo. This "go no-go" testing ensures the batch jobs submitted to the OSG, or local farm, are likely to succeed, saving a bulk of the resources for "typo free" projects likely to produce usable simulation results. If a user's project fails to test they receive an automated email containing information on the crash (stdout and stderr) as well as a custom link allowing the user to make corrective changes to the request. Once corrected, the project is automatically flagged for a retest. The entire system is supported by a database which contains information from every submitted project and submitted job.

The system itself has the ability to run on several systems automatically,

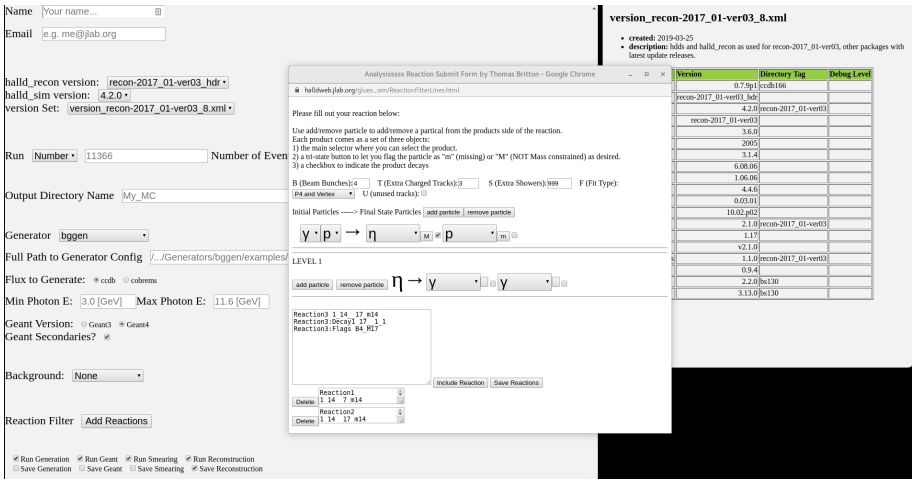

Figure 2: Shown is part of the dynamic web form which is used to submit a project to MCwrapper bot. This form includes knowledge of the GlueX software stack, a bevy of options, and integration with other GlueX systems (in the pop-up window). The system shown in the pop-up window allows users to set up reactions, in plain text, that will be searched for in the simulated data; mimicking the process real data goes through.

making the decision on which system to run on automatically on a job-by-job basis. This allows for global load balancing across multiple platforms. The system could, with additional development, dynamically aggregate jobs to target payloads specific to the systems MCwrapper-bot utilizes. Further optimizations can be achieved by leveraging the go no-go tests locally to better tailor compute resource requests for each project or job aggregations. Each submitted job is monitored in near real time. The system has some understanding of common failure modes, automatically taking appropriate corrective actions and resubmitting these jobs.

With increased usage MCwrapper and MCwrapper-bot has seen its share of scaling challenges. For example, many simulations end up needing access to "random trigger" events (a 100 Hz asynchronous trigger is used to collect hit level information from detectors allowing the use of backgrounds coming from actual data) to be merged in with pure simulation output. These files vary in size with a mean close to 1 GB. Each job needs only a slice of one of the files. The naive solution has every job take an entire copy of the needed file and proceeds to get the necessary chunk on the worker node. This, when a sufficient number of jobs are submitted, would lead to an I/O limited state on the submit node and did, in cases, saturate the entire out-bound bandwidth of Jefferson Laboratory. To reduce the load XRootD was implemented to stream parts of files to the worker node as needed. Utilizing this technology reduced bandwidth consumption by 90% and allowed for the files to be hosted separately from the

submit host.

Since its inception MCwrapper is heavily utilized collaboration wide, supplanting most, if not all means of producing Monte Carlo in GlueX. MCwrapper-bot has seen increased usage month-over-month use. At the time of writing MCwrapper-bot has been used by 48 unique users (40% of GlueX members) to produce over 52 TeraBytes of simulation with almost 1.8 million jobs which consumed almost 400 cpu-years. In the future MCwrapper-bot should utilize a more sophisticated decision making algorithm to dynamically scale and distribute elastic workflows, shuttling jobs to locations to optimize throughput, and provide multi-cluster load balancing for simulation production in Hall-D and beyond.

Progress %	ID	Email	Status	RunNumLow	RunNumHigh	NumEvents
0	810			30274	31057	10000000
0	809			30274	31057	10000000
0	807			30999	30999	10000
18.18	803			41343	41347	5000000
81.08	802			51688	51721	5000000
82.68	800			41556	41563	5000000
18.18	799			41343	41347	5000000
89.19	798			51688	51721	5000000
73.64	797			51643	51682	5000000
2	796			42559	42559	1000000
0	795			41556	41563	5000000

Figure 3: Shown is part of the dynamic dashboard (emails have been redacted to protect user information). Each request has a progress bar which updates in near real time without the need for page reloading. The Status column shows failed tests (red), successfully tested and running projects (green), and projects currently being tested (an animated ellipses). The rest of the table shows basic information about the request. The rows can be right-clicked to interact with the project (user's options are privilege dependent). Left-clicking on a row generates additional tables with more detailed information. Not shown are heart beats from the components of the automated system, the current load as seen from the submit host, and a world showing the geo-locations of the current active projects or, if a project has been selected, the scraped geo-location of the selected project's jobs.