STAR Data Production Workflow on HPC: Lessons Learned & Best Practices

M D Poat¹, J Lauret¹, J Porter², J Balewski²
¹ Brookhaven National Laboratory, P.O Box 5000, Upton, New York 11973-5000, USA
² Lawrence Berkely National Laboratory, 1 Cyclotron Rd, Berkeley, California 94720, USA

mpoat@bnl.gov, jlauret@bnl.gov, rjporter@lbl.gov, balewski@lbl.gov

Abstract. The Solenoidal Tracker at RHIC (STAR) is a multi-national supported experiment located at Brookhaven National Lab. The raw physics data captured from the detector is on the order of tens of PBytes per data acquisition campaign, which makes STAR fit well within the definition of a big data science experiment. The production of the data has typically run on standard nodes or on standard Grid computing environments. All embedding simulations (complex workflow mixing real and simulated events) have been run on standard Linux resources at the National Energy Research Scientific Computing Center (NERSC) aka PDSF. However, HPC resources such as Cori have become available for STAR’s data production as well as embedding, and STAR has been the very first experiment to show feasibility of running a sustainable data production campaign on this computing resource. The use of Docker containers with Shifter is required to run on HPC @ NERSC – this approach encapsulates the environment in which a standard STAR workflow runs. From the deployment of a tailored Scientific Linux environment (requiring many of its own libraries and special configurations required to run) to the deployment of third-party software and the STAR specific software stack, it has become impractical to rely on a set of containers containing each specific software release. To this extent, solutions based on the CERN VM File System (CVMFS) for the deployment of software and services have been employed in HENP, but one needs to make careful scalability considerations when using a resource like Cori, such as not allowing all software to be deployed in containers or bare node. Additionally, CVMFS clients are not compatible on Cori nodes and one needs to rely on an indirect NFS/DVS mount scheme. In our contribution, we will discuss our strategies from the past and our current solution based on CVMFS. Furthermore, running on HPC is not a simple task as each aspect of the workflow must be enabled to scale, run efficiently, and the workflow needs to fit within the boundaries of the provided queue system (SLURM in this case). Lastly, we will also discuss what we have learned so far about what is the best method for grouping jobs to maximize a single 48 core HPC node within a specific time frame and maximize our workflow efficiency.
1. Introduction

The STAR experiment at RHIC located at Brookhaven National Laboratory, Upton NY has been in service since the early 2000s. STAR generates 10s of PB of raw data per year that needs to be produced and analyzed. STAR has been running part of its analysis workflows as well as all of its simulation workflows on PDSF for roughly 20 years. PDSF [1] is a networked distributed computing cluster that was designed primarily to meet the detector simulation and data analysis requirements of Physics, Astrophysics, and Nuclear Science collaborations. PDSF followed standard High Throughput Computing (HTC) principles which matched other national facilities and perfectly met STAR’s needs. However, as per April 2019 PDSF has been retired. The retirement precipitated a migration plan to the High-Performance Computing (HPC) system known as Cori [8], located at NERSC in Lawrence Berkeley Lab, Berkeley, CA. While the migration to Cori may appear trivial at first, you must rethink your entire workflow when moving from HTC resources to an HPC resource as it is not just a simple remapping of jobs into this new environment. In this paper, we will discuss how Cori operates using Docker containers, CVMFS as a software distribution service, and how the STAR workflows are mapped onto an HPC paradigm as well as show some of the best practices and lessons learned.

2. Cori

Cori is an HPC Cray XC-40 Supercomputer located at NERSC and was ranked as the 10th most powerful supercomputer on June 2018 list of Top 500 supercomputers in the world [2]. Cori is made up of 2388 “Haswell” and 9688 “Knights Landing (KNL)” compute nodes. The Haswell nodes have 32 Cores (64 vCores) at 2.3 GHz per core and 128 GB of RAM equivalent to 2 GB of RAM per vCore. The KNL nodes each have a single-socket Xeon Phi Processor card installed which provides 68 1.4 GHz Cores (272 vCores) and 96 GB of RAM which equates to 0.35 GB per vCore or 1.4 GB per Core. With two different sets of hardware along with future diverse compute nodes appearing with Cori-2, one can already see how workflows need to be optimized depending on which set of nodes is being used. For instance, our STAR jobs require roughly 1.2 – 1.4 GB of RAM per job per core, therefore we could not use the vCore’s on the KNL nodes but can use vCore’s on the Haswell nodes.

A second point to keep in mind is that Cori is a shared facility used across hundreds of independent collaborations and thousands of users. Since each collaboration has different workflows and requirements, bare metal on Cori nodes cannot be configured for each use case or to the need of a specific experimental group. However, the use of containerization not only brings the flexibility, reproducibility, and usability for running scientific applications on Cori, but allows a multi-tenant approach tuned to the use of the HPC system. A system called Shifter [3] enables users to run Docker containers in a secure fashion and run containers in the SLURM batch system at scale. When containers are pulled down from Docker hub, they are converted from Docker containers into Shifter containers (a common format that can then be distributed and launched on Cori/HPC). The converted containers have some limitations such as “no root user access”, “no /var or /etc” directories as they are reserved for the system. The container also runs as “read only” i.e. you can only write to shared file systems or RAM disks mounted inside the container at start-up.

3. STAR Software in Docker

At STAR, the first approach was to start using a Docker container composed of a Scientific Linux 7 (SL7) Operating System (OS) environment and then build our software stack on top of the OS. Our initial target goal for a versatile container was to provide a base container composed of SL7 and approx. 600 RPM packages, and the ROOT [10] framework (3GB container in size). Then our plan was to take this base container and branch it out installing each necessary STAR library into its own container (SL17a, SL17d_embed, SL18b, etc… a 4 GB container by then) or creating “service containers” (for example, adding a ready-to-go MySQL service for STAR’s use).
Figure 1. This diagram is showing our initial model of a “container maintenance tree” displaying the creation of a new container for every current STAR library.

As shown in Figure 1, the container maintenance tree model would enable STAR to have everything packaged up into its own per-library container, so that can be totally independent of “remote” contents (database payload is as trivial as having a file on a centrally available file system at NERSC and pulled at runtime into the service container before the service starts). While having all software (and associated libraries) packed into its own container is convenient for use and distribution, the maintainability of this model requires the constant creation of new containers as new libraries are released throughout a given year. STAR typically releases from half a dozen to a dozen new libraries per year, so having to create a new container every time would likely add an amount of work on an already thin support team. For example, to add a new library, we would need take our base container and install the relevant software release, then commit the new container to Docker hub, pull down to Shifter on Cori, then run and test. To make matters worse, if we have many containers already deployed (each with their own software release), and if we needed to make changes to the base level of all deployed containers (i.e. install a new RPM package, etc.) then we would need to be sure the change is deployed everywhere. We quickly realized this method would be time very consuming and due to the complexity and the difficulties with managing multiple containers, we decided to take a different approach entirely and fold back to the use of a software distribution service.

4. CVMFS and Squid
CVMFS provides a read-only scalable, reliable, and low maintenance software distribution service [4]. It was developed to assist High Energy Physics (HEP) collaborations to deploy software on distributed infrastructure. CVMFS configurations are typically deployed with a Stratum 0 [11] repository server.
and Stratum 1 [12] replica servers. CVMFS is meant to be read-only for its users and the layered set of services is roughly as follows. The Stratum 0 repository is the single source where new data is written. Stratum 1 replica servers on the other hand are read-only replica’s that reduce the load on the Stratum 0. Multiple Stratum 1 servers (replicas) can be deployed, and for scalability purposes, it is wise to do so if you intend to be serving many clients. Additionally, when deploying Stratum 0/1 servers on your cluster, it is highly recommended to deploy Squid proxies that will cache the data read from the Stratum 1. Multiple Squid proxies can be deployed as well, further spreading the load. Lastly, the CVMFS clients will also cache read data as well.

**Figure 2.** This diagram shows CVMFS configuration deployed at BNL for the STAR’s Online Linux Pool Cluster. The CVMFS Stratum servers is located on the Science DMZ network zone and our Online Linux Pool is behind a protected network.

As shown in Figure 2, at BNL we had Stratum 0 and Stratum 1 servers put in place by the RHIC and Atlas Computing Facility (RACF) personnel. It is important to realize that the RACF is located on the Science DMZ network zone ("sDMZ"). Our online servers (hardware resources close to the experimental apparatus) are however located on the “Campus” network, a zone that has a higher set of Cyber protection. In order to access CVMFS as a service, we deployed our own Squid Proxy server and configured CVMFS’ clients on our Online Linux Pool (OLP) to read from the Squid Proxy which was setup to retrieve data from the Stratum 1. Our proxy node was configured as a dual networked acting as a bridge of the networks establishing fast I/O to the CVMFS clients. The proxy was setup to have 25 GB of cache which was believed (and measured) to be enough space to cache our third-party software and several versions of our libraries. Additionally, the CVMFS clients each have 25 GB of cache as well, the cache is sustained on the client disk after 1 read event. Typically, STAR runs multiple production campaigns at a time (i.e. several library versions are needed) at the RACF but target a single rather long production campaign to dedicated resources such as NERSC/Cori. Hence, the cache choice was made to essentially test, within a mini-representation of a boundary between networks, how efficiently CVMFS + squid caching would work for our “a few production campaigns at a time” use case (an over-estimate of what we would typically run at NERSC).

#### 4.1. CVMFS Testing Online

To be sure the Stratum servers, Squid Proxy, and CVMFS clients behave as intended, a real-world test was performed by running and using STAR’s library validation regression test suite (aka “nightly test jobs”) on our OLP and in a controlled environment. For this test, Squid Proxy cache, CVMFS client caches and 100 allocated slots on our OLP are cleared. To monitor how the cache is behaving, we use 30 nodes, running 3-4 jobs each.
In Figure 3, the first plot at the top is showing the Total number of Running Jobs in 10 min intervals and chunks over a 10-hour period. The jobs do not all start at the same time (a typical batch system on HTC behavior) and they all finish at different times, some finishing within the first hour and some running over the entire 10-hour period. As the number of jobs ramps up, each compute node is loading our software from CVMFS via the Squid Proxy. The second plot, showing the Squid Proxy Cache, which clearly indicates the cache begins to fill immediately. The third plot at the bottom is showing the Aggregate CVMFS Client Cache. The client cache also begins to fill up nearly in sync with the Proxy server cache fill. After 1.33 hours have passed, the Squid Proxy cache and CVMFS client cache stop increasing (STAR Software and library in other words is fully cached). Any job starting thereafter would read from the local node cache. As a side note, when new files or changes are made to the Stratum 0 server, a commit marker file is created in the filesystem which tells CVMFS to commit new changes. While the change is not immediately propagated to all clients, proxies, etc. the files will appear within minutes. This situation woud however NOT arise in STAR. Each library version constitutes a fixed release and software are not patched for deficiencies but instead, a new version is released. In other words, each library revision/version is an immutable fixed snapshot of the state of the STAR software at a moment in time and would never change after the initial release. This modus-operandi fits very well within the functional requirement of remote workflows supported via CVMFS (once cached, nothing else would ever be read over the network).

5. CVMFS on Cori
Native CVMFS mounts require access to the Linux Kernel module known as FUSE [9]. On Cori, we do not have access to the Kernel on any of the nodes, therefore the filesystem is mounted in a special way. Data Virtualization Service (DVS) servers are put in place on Cori. DVS is a network service that provides compute nodes transparent access to file systems mounted on service nodes [5]. DVS servers
forward IO via the NFS protocol. On Cori, there are 32 DVS servers, 4 dedicated to forwarding CVMFS (but not dedicated to STAR workflows alone). DVS servers are fast and do data caching of their own. However, a major drawback is that they do not maintain a Metadata cache hence, one should avoid any Unix/Linux commands that would trigger or force a Metadata lookup. For example, a login script that would perform command such as ‘ls -l’, ‘find’, or other metadata lookups when activated would cause (as many jobs may start simultaneous on an HPC system) a massive Metadata lookup as the DVS servers will have to go back to the main servers to provide it. As a result, such login script will cause significant slowdowns of the entire system. The first lesson learned is that one needs to revisit all login and software environment setup scripts in general and make sure issuing such commands is kept to a bare minimum and/or eliminated entirely. This first precaution being kept in mind, one can test how DVS/CVMFS scales as a number of jobs by measuring the time taken for diverse phases of the workflow.

5.1 Testing CVMFS on Cori/KNL

The CVMFS performance on Cori was evaluated by reserving two sets of nodes, one by reserving 50 KNL nodes for 24 hours and the second by using 300 KNL nodes for a period of 12 hours. The first reservation allowed us to test at small scale and work out the kinks in our workflow steering scripts. The larger reservation was made to execute realistic workflows that is, a more typical number of jobs for a burst production scenario and this, over a large number of nodes. With the first reservation, we started with 300 jobs on 5 nodes (60 jobs per node) and incrementally increased by 300 job 5 node blocks until we reached a total of 2400 jobs (40 nodes). On the second reservation we ran 9000 jobs (150 nodes), then 14400 jobs (240 nodes).

Figure 4. The plots above are showing two important metrics. The left plot is showing the throughput maximization defined as the number of events processed per minute per node. The right plot is showing the average time in seconds to source the login and STAR environment setup scripts defining a specific library version all done from CVMFS as we increase the number of jobs per nodes.

As seen in Figure 4, we focus to understand (evaluate) CVMFS performance on Cori while the number of jobs is increasing . The left plot shows the number of events completed per minute per job as a function of the number of jobs, increased from as little as 300 jobs up to 14,400 jobs. The first point is 300 jobs (5 nodes) and each of the 300 jobs run produced an average of 0.15 events in a minute (or a total throughput of 45 events across all jobs) with minor standard deviation. The curve shows nearly flat behaviour across all of the tests which indicates that as we increase the number of jobs/nodes reading the STAR software from CVMFS, there is no added latency (even at 14,400 jobs). The right plot is showing two curves, one shown in blue and the other in orange. The blue curve represents the average time in seconds to source the login script from CVMFS and the orange curve represents the average time to source the STAR environment (necessary to switch and load a given library version) from CVMFS. Both curves are flat for the most part with a bit of IO pile-up near the 15k job mark, however
this high point is still within the acceptable 4-6 seconds range. Overall the flat curves in the plots show CVMFS does scale on Cori and can be used to serve our software at a large scale with little to no latency.

6. STAR Workflow on Cori

A workflow to run on Cori must be carefully designed and tested as the container requirement and restrictions will demand attention before launching thousands of jobs (that may start at once). For example, to run the STAR main executable (a.k.a. “root4star”) and proceed with a real data production, we need access to our database in order to get the calibration and configuration constants. Since our focus was to test the scalability of CVMFS and not so much MySQL, we decided that for every 5 batch nodes running STAR tasks, a 6th batch node would be dedicated as a STAR MySQL Database server. This was done to ensure isolation of the CVMFS performance measurement from any other potential slowdowns and other service scalability issues. With the dedicated MySQL server node, and 60 STAR jobs per batch node, that means 300 jobs per database, which in our case is quite liberal. We kept the job to database ratio low during our tests to ensure we would not to strain the database resources causing a skew in our performance test results. However, when we run our normal production workflows, we do not dedicate an entire Cori node to run the MySQL Database server(s) (it would be wasteful). Instead, we run it as an additional process on the batch nodes alongside STAR data processing jobs.

![Steering Script](image)

**Figure 5.** Illustration of our workflow starting from the batch submission and ending at the output file. The Steering Script (left) is designed to start the MySQL DB Server Shifter containers and the DB service, make sure it is running, then launch the STAR Shifter container providing the STAR software. On the right, we show what an individual reconstruction job does via the r4s.csh script - it runs on each of the STAR Shifter containers, the actual data reconstruction process is named “root4star”.

While we set the MySQL setup aside, it is important to note that when running on such a large compute resources (such as an HPC cluster), you must use them in an efficient manner as what may appear to be a small inefficiency on one node, can be exasperated when you scale to 100s or 1,000s of
nodes. In the past, we found the most efficient number of processing jobs per node for our workflow to be 1.9 per physical cores on the Haswell (or 31 running processes on a node) and ~ 60 jobs per node on the KNL. We determined this by starting a small number of jobs based on the expected memory usage (i.e. 1 for every 2 vCore) then increasing the number until we found the best utilization of the node per hour. Best utilization in other words is defined for us as the point of maximization of the number of events processed per unit of time.

Our workflow, in a general view, is shown in Figure 5. We’re using 12 batch nodes, 10 nodes for running data reconstruction jobs and 2 for running MySQL (a small scale of what has been discussed previously). To start, we used a steering script which is submitted defining the number of nodes we want to use to run jobs, the number MySQL Databases nodes, and how many jobs per node we want to run. The steering script will first fire the MySQL Database containers which run their own process to copy the Database payload into a RAM disk, start the MySQL Database, then confirm with the steering script that all is OK. Next, the steering script runs a shell script (called r4s.csh) on each of the 10 remaining batch nodes (right portion of Figure 5). Each node will choose a random input (DAQ) file to process from a provided list, the script will sleep for a random time between 1-60 seconds before sourcing the login script and STAR environment from CVMFS. This ensures that all nodes don’t source/read from CVMFS at the same time – despite our concern to remove MetaData lookups as noted before, this remains pertinent as we scale to 10k+ jobs. Then, the script will begin executing the STAR reconstruction executable “root4star”, also sleeping for a random time between 1-60 seconds between each of the 60 jobs start time (ensuring we create an offset of when the input files are all read in reducing the load on LustreFS). Therefore, it can take anywhere from 1 minute to 36 minutes before all root4star tasks are started. This spread ensures the load is distributed and we do not pound on the system all at once. As it can be seen there are many checks before moving to a next step and multiple points of sleep delay to create IO offset in the workflow. Without such measure, it is very easy to break things on Cori and one must remain focused on the consequences of having thousands of jobs starting simultaneously and impact on the diverse sub-systems (i.e. Database, CVMFS, Lustre, … etc ..). Random time delays are easy ways to spread the load (and seconds are not consequential over the runtime of our reconstruction tasks which is of many hours long).

7. Observations & Best Practices
Before this scaling consideration, whenever creating Docker containers (to be Shifter-ized), consider if you want to install your entire software stack, RPMS, and so on into a single container or use a mechanism a-la-CVMFS for software provisioning. Self-contained (all in the container) software environments are practical if you require no other services (CVMFS, DVS, …) but will require maintenance over time if you need to commit changes regularly to either the base or release new code version (library version in our lingo) often. If your software stack rarely or never changes, then committing everything to the container may be acceptable – on the other hand, relying on CVMFS requires additional services to be started (and tested at your home institution) but this approach normalizes the installation with no other workload on the experiment’s librarian tasked to deploy new software releases.

At Cori, the use of DVS means you need some thinking to avoid causing MetaData lookups. Review and revisit all login scripts prior to running and eliminate as much unnecessary MD lookups as possible (ls -l, and the likes). Also, always consider that all jobs and tasks may start at/or near the same time. This implies a focused view on the impact of having your file system IO kicked up simultaneously, all data reconstruction tasks banging on the database at the same time or loading the dynamic libraries at the same time from 10k+ running reconstruction tasks. An easy and low-end way to spread the load is to add random times i.e. consider inserting time delays to avoid burst loads.
Another consideration before running on Cori is to find how much memory a single core job requires and understand how this will map onto a single node/core/memory profile available on the HPC cluster considered. Particularly, you need to have an estimation of how many jobs you can submit per node based on memory/CPU. However, once you can run your container on Cori and can submit to the batch system, you need to test to find the optimal number of jobs that can run on a single node. Start at a low number of jobs (or below your expected optimal point) and increase until you obtain the best utilization of the node per wall clock hour (you need to define your metric for “best utilization” – we define it as maximizing the number of Physics events produced per unit of time).

Debugging is not immediate on HPC. Consider your workflow as a three-step process, first load the experimental environment, setup the “version” of the code to be run, then run jobs. When building the workflow, it is critical for every step to be scripted, timed, and information to be logged. Here, remember many running tasks will go to the same log file so, consider printing to STDOUT not only the primary information you desire but also information such as (PID, hostname) that differentiates the provenance of the message. This will help you track down issues, see what is slowing you down, and understand how the system behaves as you scale from 1x, 10x, 100x, to 10,000x jobs.

8. Conclusion
Running on HPC at NERSC/Cori has its advantages over a standard PDSF-like cluster – instead of a number of slots we need to have occupied across the year, large amount of resources could be pre-allocated and used at once, providing a net science acceleration for those who can plan in advance. However, this modus-operandi also means that our “trickle jobs one by one to a queue system” approach cannot be immediately mapped to the job deluge an HPC cluster with a pre-allocation of resources may create. With such resources, one needs to take scaling into account when considering their workflow.

In this study, we shown that it is easy to over-use one subsystem (DVS, LustreFS, etc.) and wasting CPU time as a consequence. By creating delays randomly spread between tasks and steps in our jobs, we showed jobs scaling well at the 15k mark and efficient use of Cori. Additionally, by logging the time is takes for each step to complete such as starting the database, sourcing the environment, and time to process an event is all critical information when scaling up and finding what is slowing you down or even finding the optimal throughput for your use case. Following basic principles, STAR has run at extremely high efficiencies on Cori and in excess of 97% (see [6]) and ever improving adding and testing new components carefully. Tested independently and considering the scalable nature of CVMFS served over DVS (and its ease of use), we will be moving to a CVMFS based software provisioning model in future in support of our real STAR data production and embedding simulation workflows.

More studies are needed to find the optimum workflow for maximum use of time allocation (i.e. 48 wall clock hour time blocks) using the “convoy” model [6] and this is a task for our future contribution. As Cori-2 comes online however, it is clear that we will need to retest our workflow and find the peak-efficiency and sweet spot – we are however in the process of investigating an auto-tune model based on the type of node we are running on. Overall, we hope this work can serve as a model and lessons learned for other experiments looking to run on HPC.

Acknowledgements
This work was supported by the Office of Nuclear Physics within the U.S. Department of Energy’s Office of Science.

References
[1] PDSF, the Parallel Distributed Systems Facility operated at NERSC from 1996-2019 - https://www.nersc.gov/users/computational-systems/pdsf/about-pdsf/overview/
[2] Top 500 - June 2018 https://www.top500.org/lists/2018/06/
[3] Shifter https://docs.nersc.gov/programming/shifter/overview/
[4] CernVM File System (CernVM-FS) https://cernvm.cern.ch/portal/filesystem
[5] Sugiyama, S. and Wallace D. Cray DVS: Data Virtualization Service https://cug.org/5-publications/proceedings_attendee_lists/2008CD/S08_Proceedings/pages/Authors/16-19Thursday/Wallace-Thursday16B/Sugiyama-Wallace-Thursday16B-paper.pdf
[6] M. Mustafa, et al. - “STAR Data Production at NERSC/Cori, an adaptable Docker container approach for HPC”, CHEP 2016 conference proceedings, J. Phys.: Conf. Ser. 898 082023, doi 10.1088/1742-6596/898/8/082023
[7] Balewski, J., Porter, J., Rath, G., Lee, R., Quan, T. (2018) PDSF – Status & Migration to Cori HEPiX Fall 2018, Barcelona https://indico.cern.ch/event/708041/contributions/3276344/attachments/1812284/2960438/P ost_STAR_Data_Prod_ACAT.pdf
[8] The Cori system at NERSC, https://www.nersc.gov/users/computational-systems/cori/
[9] Filesystem in Userspace (FUSE) https://en.wikipedia.org/wiki/Filesyste_in_Userspace
[10] ROOT https://root.cern.ch/
[11] CVMFS Stratum 0 https://cvmfs.readthedocs.io/en/stable/cpt-repo.html
[12] CVMFS Stratum 1 https://cvmfs.readthedocs.io/en/stable/cpt-replica.html