Physics Data Production on HPC: Experience to be efficiently running at scale

M D Poat[1], J Lauret[1], J Porter[2], J Balewski[2]
[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 the Brookhaven National Lab and is currently the only remaining running experiment at RHIC. The raw physics data captured from the detector is on the order of tens of PBytes per data acquisition campaign, making STAR fit well within the definition of a big data science experiment. The production of the data has typically run using a High Throughput Computing (HTC) approach either done on a local farm or via Grid computing resources. Especially, all embedding simulations (complex workflow mixing real and simulated events) have been run on standard Linux resources at NERSC's Parallel Distributed Systems Facility (PDSF). However, as per April 2019 PDSF has been retired and High Performance Computing (HPC) resources such as the Cray XC-40 Supercomputer known as "Cori" have become available for STAR's data production as well as embedding. STAR has been the very first experiment to show feasibility of running a sustainable data production campaign on this computing resource. In this contribution, we hope to share with the community the best practices for using such resource efficiently. The use of Docker containers with Shifter is the standard approach to run on HPC at NERSC – this approach encapsulates the environment in which a standard STAR workflow runs. From the deployment of a tailored Scientific Linux environment (with the set of libraries and special configurations required for STAR to run) to the deployment of third-party software and the STAR specific software stack, we've learned it has become impractical to rely on a set of containers comprising each specific software release. To this extent, a solution based on the CernVM File System (CVMFS) for the deployment of software and services has been deployed but it doesn't stop there. One needs to make careful scalability considerations when using a resource like Cori, such as avoiding metadata lookups, scalability of distributed filesystems, and real limitations of containerized environments on HPC. Additionally, CVMFS clients are not compatible on Cori nodes and one needs to rely on an indirect NFS mount scheme using custom services known as DVS servers designed to forward data to worker nodes. In our contribution, we will discuss our strategies from the past and our current solution based on CVMFS. The second focus of our presentation will be to discuss strategies to find the most efficient use of database Shifter containers serving our data production (a near "database as a service" approach) and the best methods to test and scale your workflow efficiently.

1. Introduction

The Relativistic Heavy Ion Collider (RHIC) is located at Brookhaven National Laboratory (BNL), Upton NY. The STAR detector at RHIC produces 10s of PB of raw data every year which has been produced on PDSF at the National Energy Research Scientific Computing Center (NERSC) located at Lawrence Berkeley Lab, Berkeley CA for 20 years. PDSF has now been retired and we have migrated our production workflow to the High-Performance Computing (HPC) system known as Cori. In a previous work which was presented at the Advanced Computing Analysis Techniques (ACAT) 2019 conference [1], we focused on our container build model (Container Maintenance Tree), CVMFS, Squid, and an initial workflow focused on the scalability of CVMFS on Cori. In this contribution, we will discuss our new container and model for database (DB) access for STAR's data production tasks on Cori, a new data production workflow, and a concentration of our efficiency on Cori with an emphasis on maximizing events processed / second / $USD. We hope this contribution serves the community well for those who are focused on doing real physics data production on HPC.

2. Cori

Cori is an HPC Cray XC-40 Supercomputer located at NERSC and is now ranked as the 13th most powerful supercomputer on the November 2019 list of Top 500 supercomputers in the world [2]. While Cori is continually upgraded, as of now, the supercomputer is composed of 2388 "Haswell" nodes with 32 Cores (64 vCores, with 2-way Hyperthreading) and 128GB of RAM and 9688 "Knights Landing (KNL)" nodes with 68 Cores (272 vCores with 4-way Hyperthreading) and 96 GB of RAM. When STAR runs its software, each STAR task requires 1 thread and 1.2 – 1.4 GB of RAM. On Cori, we evaluated both the KNL & Haswell nodes to find the maximum number of STAR tasks that could run on each node.

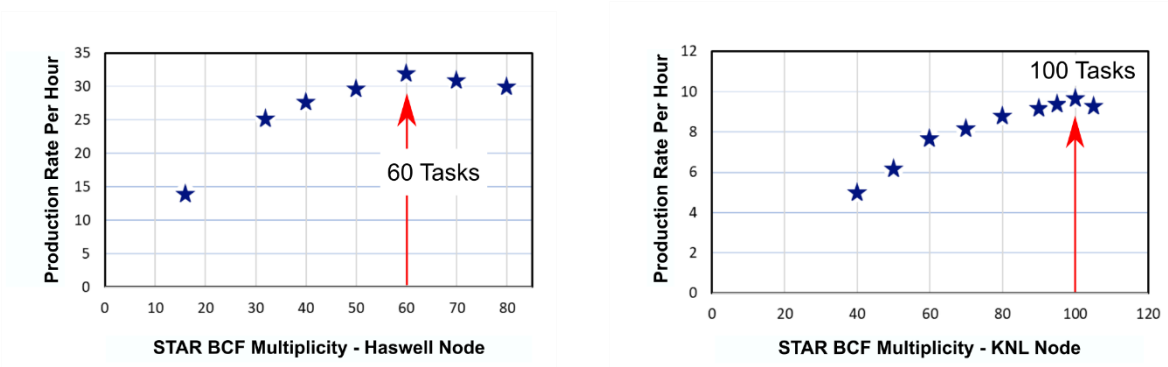

Figure 1. The two plots are showing the number of STAR tasks, known as ROOT4STAR (R4S), being produced per hour as the number of parallel tasks is increased. The left plot is the test ran on Haswell while the right plot is the test KNL. The x-axis is the total number of parallel tasks running, the y-axis is number of hours to complete the tasks.

As shown in Figure 1, both the Haswell and KNL have a high point showing where we reach the most efficient number of parallel tasks before the production rate per hour begins to plateau. This simple baseline test gives us a starting point to see how many tasks to run per node and where to keep it consistent when scaling across multiple nodes.

2.1 STAR Container on Cori

In our previous work, we found that it was inefficient to build out our container to house a Scientific Linux 7 (SL7) environment, all our RPM, the STAR third party software stack, and a STAR library inside one container. This model was quickly proven to be a poor method for maintaining containers as any changes or maintenance to the baseline container would require all containers with a STAR library

to need updating. We have thus moved forward with the use of CVMFS [3] as a software provisioning layer to provide software deployment to a slimmed down container which only contains SL7 and our needed RPM packages. Furthermore, with our focus on efficiency and our database access, we have provided a new container with the latest MySQL version enabling STAR to run our latest data production on Cori with ability to combine the R4S tasks and MySQL server inside the same container.

2.2 CVMFS on Cori

CVMFS uses the FUSE [4] kernel module to mount natively to a Linux machine. On Cori, we do not have access to the kernel (cannot add FUSE module), therefore CVMFS is delivered to the system using Cray's Data Virtualization Service (DVS) [5]. DVS servers forward the I/O from CVMFS to Cori nodes, however the DVS servers do not support metadata lookups, meaning any such lookup requires the DVS server to look it up in CVMFS (again and again), then provide it to you. Metadata lookups to CVMFS via DVS can cause significant latency. While we are generally fine with the DVS system, for STAR to achieve good throughput maximization to CVMFS we had to modify our workflow by implementing random sleep delays for each node sourcing the STAR software from CVMFS. The sleep delays create a load spreading effect allowing us to scale, otherwise without the delays CVMFS on Cori has been known to create latency and potentially crash.

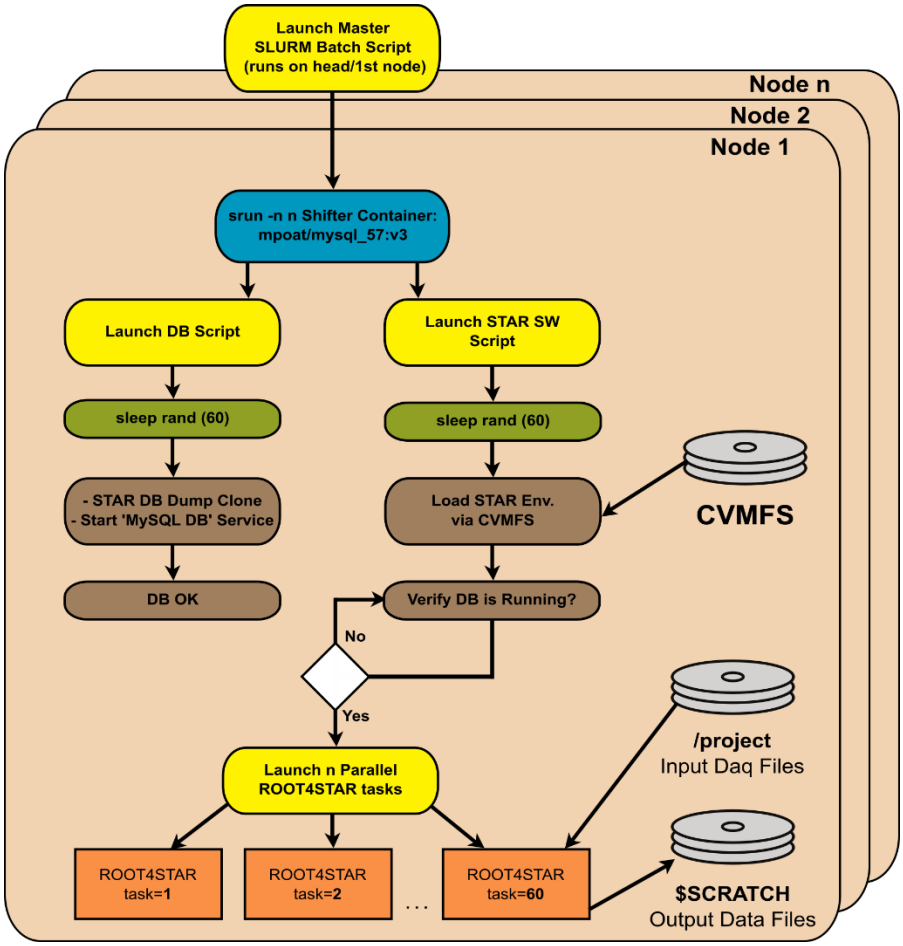

Figure 2. Diagram depicting our new workflow which starts at the Master SLURM Batch script, followed by launching the MySQL DB + R4S container and two parallel scripts (Load DB script & STAR Software Script). Once the DB is verified it is running from the STAR Software Script, we then launch parallel R4S tasks on the node.

3. STAR Workflow on Cori

With our new focus on how we provision MySQL DB alongside R4S tasks on a single Cori node and emphasising maximum efficiency, we have revamped our workflow to run on Cori to address these matters.

For simplicity in Figure 2, we are depicting the workflow as if each node runs its own MySQL DB instance. The Master SLURM Batch script is run on the head node/1st Cori node, the script sets parameters and creates needed output directories, then launches two scripts in parallel to launch the MySQL DB and the STAR Software Script. The MySQL DB script dumps the latest version of the STAR MySQL calibration DB into a node-local filesystem [6] (NERSC provides an option for building a separate XFS [7] local filesystem at runtime on each node). Once the DB payload is copied, MySQL DB server starts up, optimizes itself and will continue running as long as R4S tasks are running or the SLURM job is killed. Meanwhile, the STAR Software Script will setup its environment via CVMFS and has its own mechanism to verify the DB is up and running. Once confirmed, the script will then launch a specific number of R4S tasks (60 for Haswell, 100 for KNL) each with its own number of events per task using event parallelization across Data Acquisition (DAQ) Files.

4. Database Server on Cori Batch Nodes

MySQL Database access is required for the STAR Software to run. While STAR does have public facing DB servers at BNL that do scale, the Cori worker nodes are on an internal network and cannot access BNL DB servers. In the past, we would dedicate 1 head node on Cori to run the STAR database server serving X worker nodes. The 'head node' model as we put it creates a total efficiency loss as we dedicate an entire node to run the DB only, while all its remaining CPU cores sit idle.

At STAR we are now running MySQL 5.7 for our calibration databases. Since we intend to do real data reconstruction on Cori with new data, a new container supporting MySQL 5.7 was required. We deployed a new version of our docker/shifter container built on SL7 + all RPM libraries for the STAR software and installed MySQL 5.7 server. We then created a new snapshot of the STAR calibration databases and copied them over to a Cori project managed global filesystem. We then created special authorization and authentication mysql tables to enable the new DB to run on Cori in the shifter container. Once completed, Cori nodes could then run MySQL server and R4S in the same container on the same node.

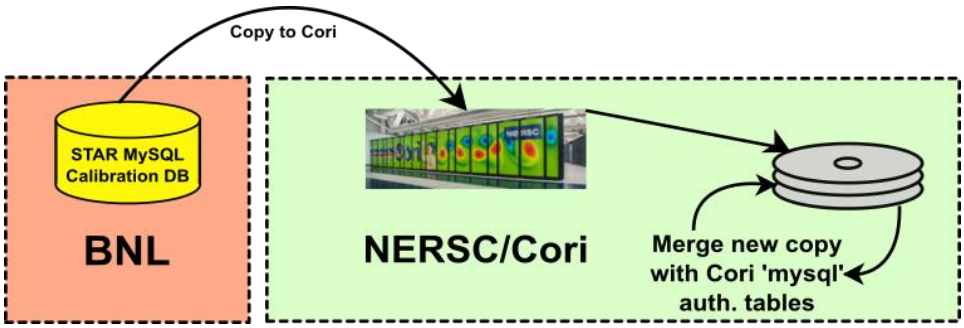

Figure 3. Diagram illustrating the STAR MySQL Calibration DB copied to Cori. The transfer is secure as it is stripped of account/password information, we then merge the MetaData payload with Cori's 'mysql' authentication/authorization table.

Once we worked out the kinks in creating a DB snapshot, copying it to Cori, and sorting out the authentication/authorization tables, we streamlined the process so that we could have hours old snapshots copied to Cori at any time. One thing to note is once we created the Cori authentication / authorization tables, they do not need to be re-created with every snapshot. The tables can simply be remerged with a new snapshot from STAR and immediately be run on Cori for STAR jobs.

5. STAR Efficiency on Cori

Our goal on Cori is to maximize our events processed / second / $USD. When you allocate time on a Cori node you pay for the actual time used on the node. Once the node is allocated the clock starts ticking, and since we are paying a real cost, we need to maximize how efficient we are on the node.

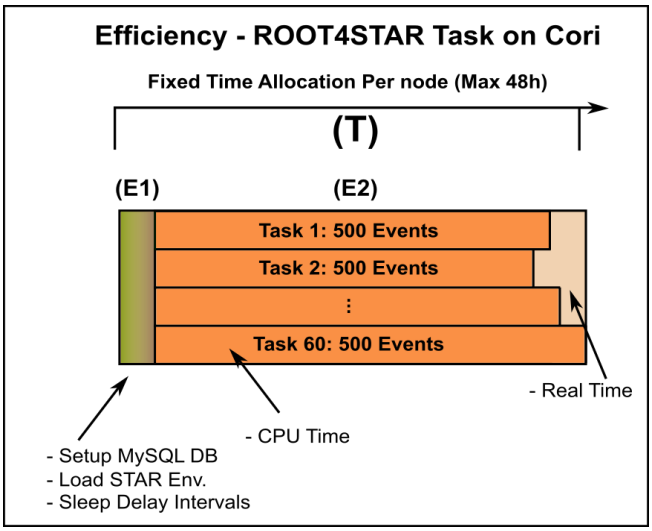

Figure 4. Illustration describing total efficiency of R4S tasks on Cori. The green/brown portion on the left is the "Job Start Efficiency" which is composed of the MySQL DB setup, loading of the STAR software environment, and sleep delay intervals. The middle portion is the CPU/Real time "Event Efficiency". The right most portion on the right is idle CPU time after a task has finished earlier than others.

In Figure 4, each portion of the workflow is shown over a fixed time allocation in terms of efficiency. When a node is allocated our scripts begin and start initial tasks such as copying the database payload to the node's virtual RAM disk and loading the STAR environment from CVMFS. Since we are measuring efficiency, for simplicity, we call this our Job Start Efficiency (E1) measuring how efficient our setup is before we can start data reconstruction tasks. Once the job starting tasks are complete, R4S data reconstruction tasks start running. R4S itself has its own Event efficiency (E2) which is the measurement of the CPU/Real time ratio of the tasks. The performance of the tasks will hinder on the MySQL database and the IO of the filesystem for inputting raw data and outputting reconstructed files. Lastly, we define our "Total Efficiency" to be the ((NodesUsed/NodesUnused) * E1 * E2) of an entire job from SLURM job start to the last task finished.

Job	(T) DB dump, Load Env., Rand (1-60s) delays	Job Start Efficiency (Total Job Time - (T))/Total Job Time (E1)	Event Efficiency All Events (E2)	Total Efficiency (NodesUsed/Nodes Unused) * E1 * E2
KNL 1 Node (Long Test - 60 task)	819 sec.	99.50%	99.79%	**99.30%**
KNL 11 Nodes 1 Node ded. DB server (60 task)	864 sec.	99.48%	99.90%	**89.44%**
Haswell 1 Node (Long Test - 60 task)	378 sec.	99.76%	99.04%	**98.80%**
BNL RCF Job - 100 tasks	1 sec.	99.99%	99.81%	**98.82%**

Figure 5. A table showing a breakdown of each efficiency as a percentage when running on KNL, KNL with a dedicated DB head node, a Haswell node, and the BNL BCF.

As shown in Figure 5, the efficiencies described earlier is shown as a percentage on both the Cori nodes and the BNL RHIC Computing Facility (RCF) pool. Overall, our Job Start Efficiency is at worst 99.5% (a 0.5% loss on KNL) and our Event Efficiency is at worst 99.04% (a 0.96% loss on Haswell). The most significant impact to the Total Efficiency has come from the change in our workflow and MySQL DB setup. When dedicating a node on Cori as a head node to run the MySQL Database server serving 10 worker nodes (1 node to 11 nodes) the Total Efficiency of this model is 89.44%, this is how we ran our workflow previously. When combining MySQL DB and STAR tasks on the same node, the Total Efficiency is 99.30%. Obviously, it is better for the worker nodes to self-serve the DB as the gain is significant, however this model does come with its own costs. As discussed earlier, the DB payload is copied into a node-local filesystem. At ~30GB of DB files copied per node, if we run on 100 nodes in parallel this would equate to 3TB copied at the start of the job. Therefore, at a large scale we would likely run 1 worker node with both MySQL DB and R4S tasks while this 1 node serves 5-10 other worker nodes in batches. This would appear to repeat the 1-to-11 model shown before, but instead head node would run both MySQL and R4S tasks. We have done preliminary tests on this model and we did not observe any scaling problems when serving up to 10 nodes but have not tried beyond that. The impact to the MySQL+R4S worker node serving out DB connections to other nodes.

5.1 Idle CPU Problem

Our last point focusing on efficiency would refer you back to the right most portion of Figure 4 showing idle CPU toward the end of the tasks. If you request 24 hours for a SLURM job on a Haswell Cori node to run 100 R4S tasks at 500 events per task and the entire job only runs for 18 hours, you will only be charged for 18 hours of use. Since each of the 100 tasks in the SLURM job is independent from each other, you are charged for the time of the longest running task (i.e. when the node is released back). What we found is that for any given STAR job (such as the example just described), the first task may finish at the 15.5-hour mark, the 50th task may finish at the 17-hour mark and the 100th task at the 18-hour mark. Therefore, towards the end of the job, more and more of the CPUs sit idle. At this time, we have not solved this idle CPU problem, however designing a throughput estimator (described in the next section) would give us a starting point on the length of a job and introducing an "event service" would enable us to launch new tasks onto the idle CPUs with low numbers of events to maximize the allocated time slot. On the other hand, if we were to over-subscribe a Cori node by not allocating enough time for the job (i.e. request 12 hours when 18 is really needed) and if our tasks do not finish gracefully from

this job, the data that is produced is not easily usable. For this, we would need to implement proper signal handling methods to ensure we can soft-kill the tasks without any data loss.

6. Throughput Estimator

To better tackle the problem of maximum CPU usage, we created a throughput estimator that will enable STAR to estimate how long a job will take on Cori. By obtaining the average time that events are processed from multiple tests ran on a single KNL node, Haswell node, and on the BNL RHIC Computing Facility (RCF).

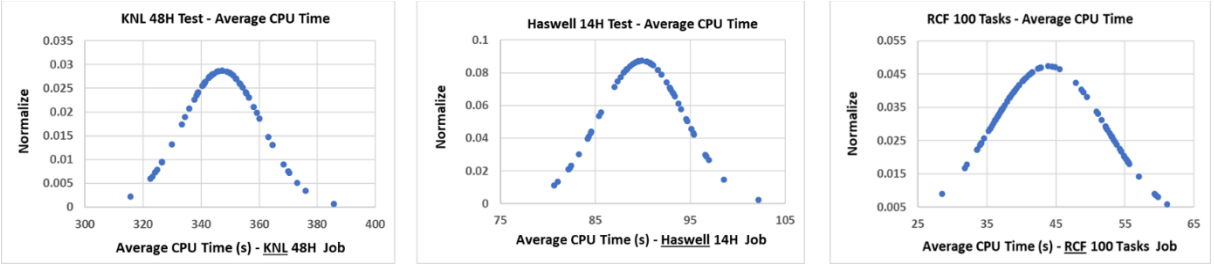

Figure 6. The three plots show the Average CPU time to process events as a gaussian curve consisting of multiple R4S tasks ran on a Cori KNL node, Cori Haswell node, and the RCF. The x-axis is the Average CPU Time shown in seconds.

As shown in Figure 6, the RCF cluster is a condor pool where each task has its own 2.8GHz Intel CPU. On the Cori Haswell node most of the tasks (60) are using hyperthreading and on the Cori KNL nodes the tasks (100) are not using hyperthreading. Irrespective of the nodes, the curve of each plot follows a gaussian where most of the time to process events falls in the center of the plot with even tails for events processed faster or slower. What this throughput estimate will provide is a starting point to create an event service to launch new tasks when one finishes. Additionally, we can use this to run any dataset on our RCF cluster at BNL (no charge to STAR like on Cori) and estimate how long it will take on either Haswell or KNL at Cori.

7. Conclusion

With our new container's ability to run MySQL DB and R4S tasks to either self-serve DB connections and serve connections to other worker nodes, we have found our most efficient workflow model to date on Cori. As new data is produced, we can rapidly produce a new DB snapshot, transfer to Cori, remerge with authentication tables and be ready to produce that new data right away. Our workflow has been tuned to parallelize the initial job start setup and maximize our total efficiency. CVMFS on Cori via DVS has been a workable solution but has required us to implement time delays during setup to spread the load.

As we discussed, the head node model of dedicating 1 Cori node to run MySQL DB alone introduces the biggest efficiency loss. Looking ahead, we plan to implement the ability to gracefully terminate tasks with the use of "signal handling", an "event service" to launch new tasks after one finishes, and the potential use of the Cori burst buffer [8] to pre-stage DB content. Overall, our total efficiency from setup to last task finish is in an acceptable range (loss <1%).

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

References

[1] Poat, M. D., Lauret, J., Porter, J., Balewski, J. (ACAT2019 Proceedings, to be published) https://indico.cern.ch/event/708041/contributions/3276344/
[2] Top 500 – November 2019 https://www.top500.org/lists/2019/11/
[3] CernVM File System (CernVM-FS) https://cernvm.cern.ch/portal/filesystem
[4] The FUSE Module - https://www.kernel.org/doc/Documentation/filesystems/fuse.txt
[5] Cray Data Virtualization Service (DVS) https://pubs.cray.com/content/S-0005/CLE%206.0.UP05/xctm-series-dvs-administration-guide/introduction-to-dvs
[6] Temporay XFS Files for Optimizing IO https://docs.nersc.gov/programming/shifter/how-to-use/#temporary-xfs-files-for-optimizing-io
[7] The XFS Filesystem https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/storage_administration_guide/ch-xfs
[8] Cori Burst Buffer - https://docs.nersc.gov/filesystems/cori-burst-buffer/