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

Michael D. Poat, Jérôme Lauret, Jefferson Porter, & Jan Balewski
Outline

▶ Introduction
▶ Containers & CVMFS
▶ STAR Data Production Workflow
▶ Database Access
▶ Efficiency & Throughput Considerations
▶ Conclusion
Introduction

- The Relativistic Heavy Ion Collider (RHIC) is located at Brookhaven National Lab (BNL) in Upton, NY
- The STAR detector at RHIC produces 10s of PB every year and ran its data production on NERSC/PDSF for ~20 years
- PDSF’s is EOL -> migrated to NERSC/Cori
- Our previous work focused on the scalability of CVMFS serving STAR Software on Cori
  - ACAT 2019 “STAR Data Production Workflow on HPC: Lessons Learned & Best Practice”, M.D. Poat et al 2019
- Current Focus:
  - Workflow
  - MySQL Database access
  - Efficiency
NERSC – ‘Cori’ Cray XC-40 Supercomputer

- 20 TB $SCRATCH/user (Luster FS)
- 2388 Xeon "Haswell" nodes
  - 32 Cores (64 vCores, 2-way HT)
  - 120 GB RAM (~ 1.8 GB / vCore, plenty for STAR)
- 9688 Xeon Phi "Knights Landing" nodes (KNL)
  - 68 Cores (272 vCores, 4-way HT)
  - 96 GB RAM (0.35 GB / vCore or 1.4 GB / core)

STAR Task Density

- Evaluated KNL & Haswell maximum utilization with STAR tasks
- STAR SW requires ~1 GB RAM
- Haswell: Supports 60 STAR tasks per/node
- KNL: Supports 100 STAR task per/node

STAR Software in Containers

- Docker/Shifter containers are required to enable the STAR Software to run on Cori
- STAR Docker containers are built based on Scientific Linux 7 (SL7)
  - SL7 + RPM (650 MB)
  - SL7 + RPM + STAR SW (3 GB)
  - SL7 + RPM + STAR SW + 1 STAR Library (4 GB)

- **Cons**: If we have to update the Base image, all images will need to be updated -> maintenance nightmare
- **Pros**: All Software and libraries packed in 1 container
- **Decision (standard practice)**: Use CVMFS for all Experiment stack related software -> standard way for software provisioning
CVMFS on Cori

CVMFS on Cori
- CVMFS requires FUSE kernel module to mount natively
- Fuse restriction on Cori (No Kernel access on worker nodes)
- NERSC provides Cori with Data Virtualization Service (DVS) servers
  - Used for I/O Forwarding and data caching
- Cori has 32 DVS Servers, 4 dedicated to forwarding CVMFS I/O
- DVS servers forward I/O well, but do not support metadata lookups (requires lookup to real CVMFS backend -> latency)

Throughput Maximization for CVMFS
- Looked at average of events produced min/“task”
- Scaled from 1 – 240 nodes
- Drops by ~10-12% at first but we still gain in “events min/node”
- Curve remains flat afterward up to our max @15,000 tasks on 240 nodes
- In order to achieve this we needed to modify our workflow with time delays…
First we launch steering script to the batch system
STAR Workflow on Cori

- First we launch steering script to the batch system
- Starts the STAR+mysqld container
- Runs ‘Load DB’ & STAR SW scripts in parallel
First we launch steering script to the batch system

- Starts the STAR+mysqld container
- Runs ‘Load DB’ & STAR SW scripts in parallel

- Both scripts have random sleep delays (one for copying the DB and 1 for loading SW via CVMFS)
- Once STAR SW is loaded the script will wait until the DB has started (biggest time killer!)
First we launch steering script to the batch system

- Starts the STAR+mysqld container
- Runs ‘Load DB’ & STAR SW scripts in parallel

- Both scripts have random sleep delays (one for copying the DB and 1 for loading SW via CVMFS)
- Once STAR SW is loaded the script will wait until the DB has started (biggest time killer!)

- Node(s) will launch ‘n’ Parallel ROOT4STAR tasks
First we launch steering script to the batch system

- Starts the STAR+mysqld container
- Runs ‘Load DB’ & STAR SW scripts in parallel
- Both scripts have random sleep delays (one for copying the DB and 1 for loading SW via CVMFS)
- Once STAR SW is loaded the script will wait until the DB has started (biggest time killer!)
- Node(s) will launch ‘n’ Parallel ROOT4STAR tasks

Job start efficiency loss

Efficiency - ROOT4STAR Task on Cori

Fixed Time Allocation Per node (Max 48h)

(T)

Task 1: 500 Events
Task 2: 500 Events
...
Task 60: 500 Events

- Real Time
- Setup MySQL DB
- Load STAR Env.
- Sleep Delay Intervals
- CPU Time
MySQL Database Access is required for the STAR Software to run

- STAR does have public facing DB servers that do scale, but Cori worker nodes are on an internal network.
- Hours old snapshots of the DB can be copied to run locally on Cori at anytime.
- Once copied, a Cori authentication table is merged with the new DB and we are ready to run.
Database Server on Cori Batch Nodes

MySQL Database Access is required for the STAR Software to run

- STAR does have public facing DB servers that do scale, but Cori worker nodes are on an internal network.
- Hours old snapshots of the DB can be copied to run locally on Cori at anytime
- Once copied, a Cori authentication table is merged with the new DB and we are ready to run

How we run the DB

- In the past, we would dedicate 1 head node on Cori to run the STAR Database serving X worker nodes
- We now have our ‘mysqld’ DB server installed in the same docker container running the STAR Software on Cori -> each node serving itself

Can worker node running DB + R4S tasks serve DB to itself & other worker nodes?

- With configuration tuning a worker node can run DB + R4S tasks to serve itself & 10s of other worker nodes
  - Default configuration DB could only handle 150 connections
  - ‘Head node’ model sacrifices an entire node

How does this affect our efficiency…?
Efficiency on Cori
Efficiency on Cori

- **Job Start Efficiency**: Real time to copy/start DB, load env., sleep delays (E1)
- **Event Efficiency**: CPU/Real time ratio for STAR event data reconstruction (E2)
- **Total Efficiency**: SLURM job Start -> Last Task Finished
  
  \[
  \text{Total Efficiency} = \frac{\text{NodesUsed}}{\text{NodesUnused}} \times E1 \times E2
  \]
Efficiency on Cori

**Goal:** Maximize (event per sec. / per $)
- NERSC charges the same for KNL or Haswell
- Dedicating 1 head node as DB only to serve 10 worker nodes (1-to-11) **VS.** (1-to 1) model (each worker node self-serves DB)
  - 1-to-1 model: Total Eff. 99.30%
  - 1-to-11 model: Total Eff. 89.44%
- **Better to self-serve DB**
- Job Start Efficiency: we lose ~.05%
- Event Efficiency: ~98-99%
- Total Efficiency on 1-to-1 KNL/Haswell, and BNL BCF: ~98-99%
- Total vCore Utilization:
  - Haswell: 87% @ 60 task + 1 DB
  - KNL: 36.9% @ 100 task + 1 DB
  - Cannot maximize CPU util. due to memory limit -> **Best to focus on packing best # of tasks per/node & Total Efficiency**

- **Job Start Efficiency:** Real time to copy/start DB, load env., sleep delays (E1)
- **Event Efficiency:** CPU/Real time ratio for STAR event data reconstruction (E2)
- **Total Efficiency:** SLURM job Start -> Last Task Finished (NodesUsed/NodesUnused) * E1 * E2

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

- When a job is submitted with multiple tasks, each task will finish at different times.
- If no new task is assigned, the CPU will sit idle
  - You pay for the total time of the longest running task
- If we push the tasks to run past the 48h time limit, **and** if it does not finish gracefully = **Data not easily usable**
- Average Idle CPU Loss at end of ~48h job
  - KNL: 0.02% CPU Time Loss
  - Haswell: 0.01% CPU Time Loss
- To Fix this “Problem” we need
  - A “Throughput Estimator” to estimate how long a job will take
  - “Signal Handling” to ensure a task can be “soft killed” properly with no data loss
  - An “Event Service” to launch new tasks
    - “Event Service” would also serve to launch new tasks with low events to maximize 48h time slot
Due to the ‘Job Start’ efficiency loss, it is best to run for the maximum amount of time (48h)

By obtaining the average time events are processed per task, we can estimate how long a job will take

- Multiple tests run on a single KNL node, a single Haswell node, & BNL RCF (2.8GHz Intel)

The distribution and scaling is very predictable between the systems on any dataset

- With the estimator, we only need to run a small batch of jobs on our BNL RCF farm to get estimate of total time on Cori KNL/Haswell

Provides starting point for “Event Service” to launch new tasks when one finishes
Conclusion

Database:

- DB can be copied to NERSC on demand and remerged with authentication tables
- On Cori: Worker node running ‘mysqld’ DB instance + R4S tasks to self-serve & serve DB connections to some worker nodes - > most efficient model

Workflow:

- Launch DB & environment scripts in parallel
- DVS for CVMFS is a workable solution but required us to implement time delays (latency)

Efficiency:

- Events produced min/node:
  - **Haswell**: 40.55 total events per min (60 tasks total)
  - **KNL**: 13.7 total events per min (100 task per node)
- Head node model introduces biggest efficiency % loss
- Haswell provides best CPU power / $ for us

Our next steps

- Ensure graceful termination of the tasks (use of “signal handling”)
- Potential use of Burst Buffer to pre-stage DB content
- “Event Service” is coming soon
- Efficiency loss at start & end of job is minimal - > acceptable range
Thanks!
Summary Slide

- Docker/CVMFS
  - Containers are kept to minimum -> SL7 + RPM + mysql
  - Software provisioned from CVMFS via DVS servers on Cori

- DB Access
  - STAR DB snapshot dumped at Cori, remerged with auth tables, then run in container to serve STAR tasks
  - Each node on Cori can run its own copy of DB + ROOT4STAR tasks & serve other worker nodes
  - Burst Buffer may be a solution to pre-stage DB copies before start of job

- Workflow: Maximize our “Job Start Efficiency” with parallel setup scripts
  - Delays for DB dump and loading software via CVMFS -> needed to not overload subsystems

- Efficiency: “Job Start Efficiency” and “Idle CPU Problem” have minimal impacts on “Total CPU/Real time Efficiency” if we run for maximize node allocation (48h)

- Places where we lose CPU time are understood – solutions underway

- Total CPU/Real time Efficiency on Cori with 1-to-1 DB model: ~98-99%