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

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

NERSC Users Group SIG Annual Meeting
August 17th, 2020
Introduction

- 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

- Ongoing Efforts for STAR Data Production on Cori
  - Container Model
  - Scalability of CVMFS serving the STAR SW on Cori
  - Workflow on Cori
  - MySQL Database access
  - Efficiency
Docker/Shifter containers are required to enable the STAR Software to run on Cori.

Best to deploy minimal containers, with Software stack provisioned from CVMFS.

Initial Container Model:
- Base OS SL7 + RPM + STAR SW + 1 STAR Library (4 GB)

Minimal Container Model:
- Base OS SL7 + RPM
- CVMFS Serves: STAR SW + STAR Libraries

Our previous setup required 1 node to run a MySQL DB container while all other worker nodes would run STAR tasks.

The current running setup combines STAR Tasks & MySQL Database on 1 node.

Current Container: SL7 + RPM + mysqld
CVMFS on Cori

Throughput Maximization for CVMFS

- Looked at average of events produced min/“task”
- 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...

- Fuse restriction on Cori (No Kernel access on worker nodes)
  - cannot mount CVMFS natively
- NERSC provides Cori with Data Virtualization Service (DVS) servers
- DVS servers forward I/O well, but do not support metadata lookups (requires lookup to real CVMFS backend -> latency)
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. This script starts the STAR+mysqld container and runs 'Load DB' & STAR SW scripts in parallel. Both scripts have random sleep delays (one for copying the DB and one 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.
Efficiency on Cori

**Goal:** Maximize (event per sec. / per $)

- 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
  
  \[
  \text{Total Efficiency} = \frac{\text{NodesUsed}}{\text{NodesUnused}} \times E1 \times E2
  \]

<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>
Conclusion

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

- **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
  - Time delays required (latency) for CVMFS via DVS

- **Efficiency:**
  - “Job Start Efficiency” and Idle CPU at the end of job have minimal impacts on “Total CPU/Real time Efficiency” if we run for maximize node allocation (48h)
  - 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
Thanks!
Due to the efficiency loss at the start & end of a job, 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 an “Event Service” to launch new tasks when one finishes.