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

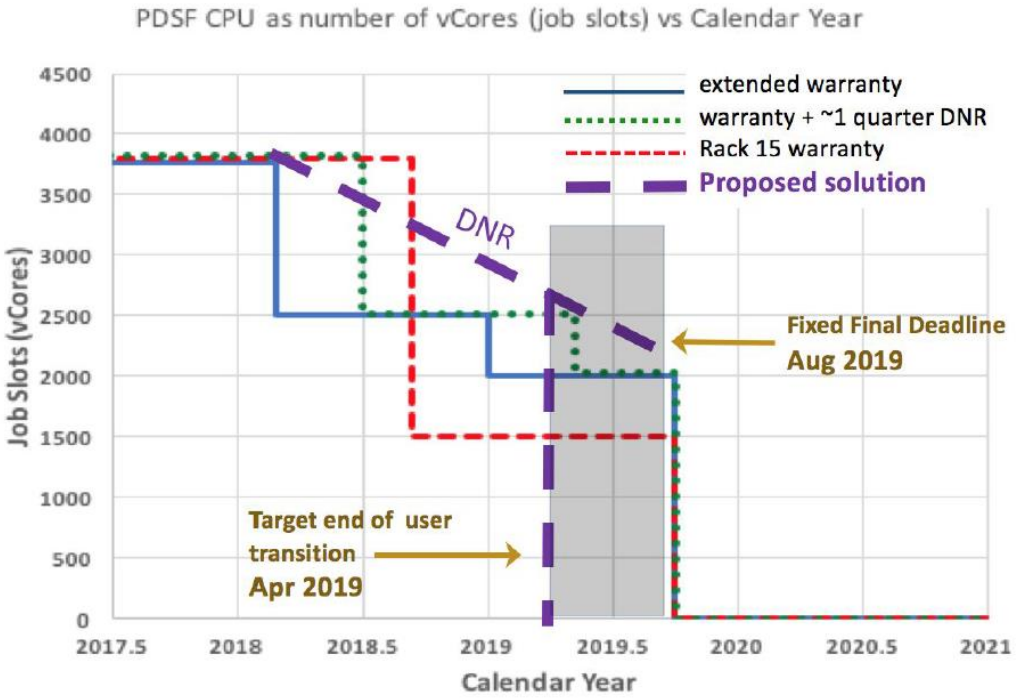
Poat, M.D., Lauret, J., Porter J., & Balewski, J.

Outline

- Introduction / Motivation
- STAR Software in Docker/Shifter
- CVMFS & Squid + Testing Online
- STAR Data Production Workflow
- CVMFS On Cori
- Best Practices

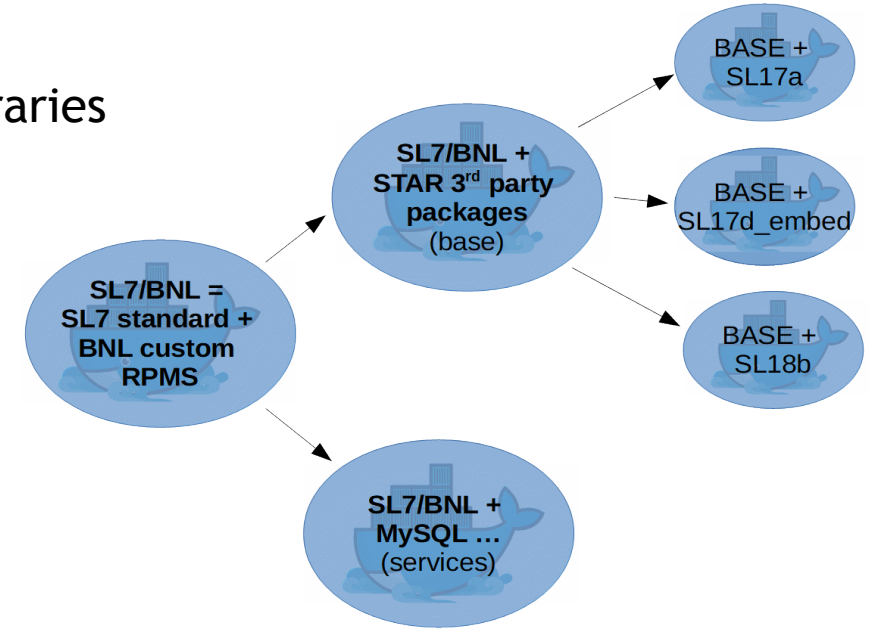
Introduction/Motivation

- The STAR experiment at RHIC has been running its data production on PDSF for ~20 years
  - PDSF is EOL and is being retired April 2019
    This precipitated a migration plan to NERSC/Cori
  - Started with a container for STAR Software including all dependencies & libraries + central DB service
  - Outcome: 25,600 cores use "emergency" production for conference, event parallelization. Presented ! CHEP 2016 **"STAR Data Reconstruction at NERSC/Cori, an adaptable Docker container approach for HPC,** M. Mustafa *et al* 2017 *J. Phys.: Conf. Ser.* **898** 082023
  - Multiple change of Cori resources – need a more agile approach
- Use of CVMFS to distribute software & libraries
  - Single point for maintenance i.e. can serve local and remote (Cori) needs
- This talk - Best practices for STAR running on Cori
  - Maintainability
  - Scalability w/ CVMFS, how-to



STAR Software in Docker

- STAR Docker container is built using 'Docker import' method
  - Create temp directory and copy needed files + install ALL rpm into temp directory -> tar into docker import
- Initial Docker containers were SL7 + rpm + STAR SW + STAR Libraries
  - SL7 + RPM (650 MB)
  - SL7 + RPM + STAR SW (3 GB)
  - SL7 + RPM + STAR SW + 1 STAR Library (4 GB)
- Pros: All Software and libraries packed in 1 container
- Cons: Maintainability cumbersome – adding new libraries, updates, and packages means creating a new container
  - (make changes to container, deploy new container to hub, pull down to shifter = time investment)



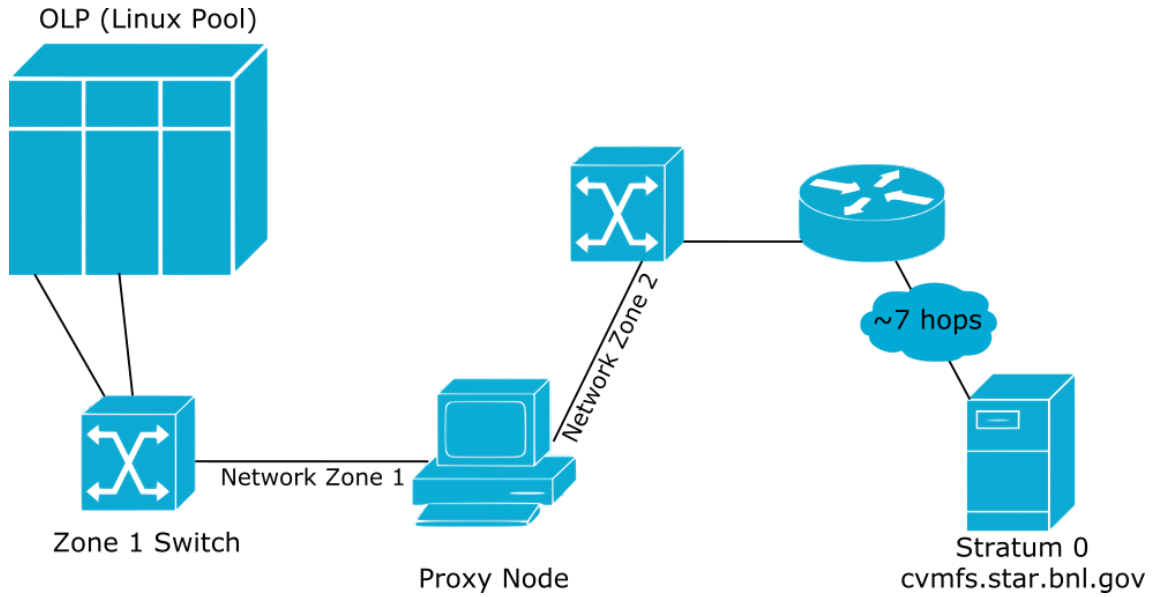
Container Maintenance Tree

- Decision (standard practice): use CVMFS for all Experiment stack related software, only base OS components are in the Docker container

CVMFS & Squid

- CERN VM File System (CVMFS) provides a read-only scalable, reliable and low-maintenance software distribution service
  - CVMFS requires tiered “Stratum” servers for reading/writing data. STAR requested BNL’s Facility to deploy Stratum 0/1 CVMFS servers
  - Client side needs CVMFS rpm + configuration requires multiple public keys (BNL & `star.sdcc` key) and config. Files (CVMFS mounts using the Fuse module, can be mounted with AutoFS)
- A Squid Proxy was deployed to support our 240 slot Online Compute Farm
  - Multiple Squid proxies recommended, we deployed only one for our small farm (for now) -> Used to reduce load on Stratum servers
  - Proxy node configured **to bridge network zones** i.e. “online” and “offline” are separated into zones. This ensures fast IO to CVMFS clients.
- Caching
  - Squid Proxy has 25 GB cache (`cache_dir ufs /var/spool/squid 25000 16 256`)

    25 GB -> Enough space to avoid churn even with many more libraries and packages added
  - Clients have 25 GB of cache stored on local disk
  - Cache data is sustained on client disk after 1 read event
- Base Software Stack and set of libraries installed in `/cvmfs/star.sdcc.bnl.gov`



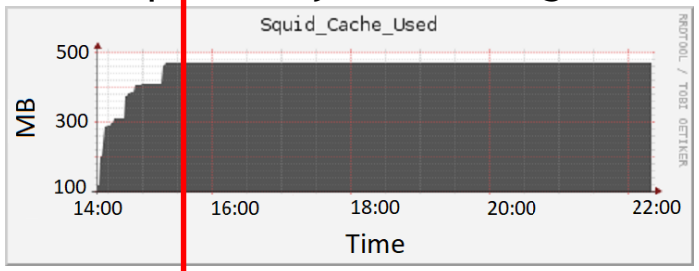
CVMFS Test Online

- Allocated 100 slots
- Cleared Squid Proxy and CVMFS Client caches
- Data production manager submitted jobs reading STAR software from CVMFS
- As jobs ramp up, cache fills on both CVMFS clients and Squid Proxy
- Squid proxy obtains IO from stratum 1 servers
- After all clients read software files, cache is kept on disk and no longer read from proxy
  <~1 GB stored
- After 1.33 hours (15:40) Squid Proxy and all CVMFS clients have Software cached, clients will read from local copy where “Squid” and “CVMFS Client” curves shows flat.
- Test #1: run online jobs via CVMFS showed squid proxy pulling from Stratum 1, serves all data to clients once i.e. everything is cached – but jobs do not start at the same time in this setup (a major difference with our HPC use)

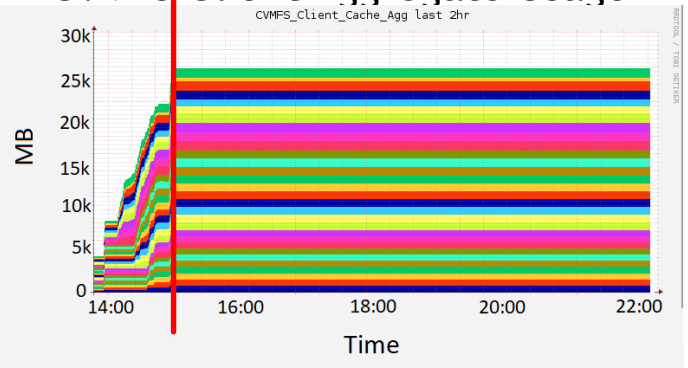
Total Jobs Running per 10 min chunks

1.33hr passed @15:20

Squid Proxy Cache Usage



CVMFS Client Aggregate Usage

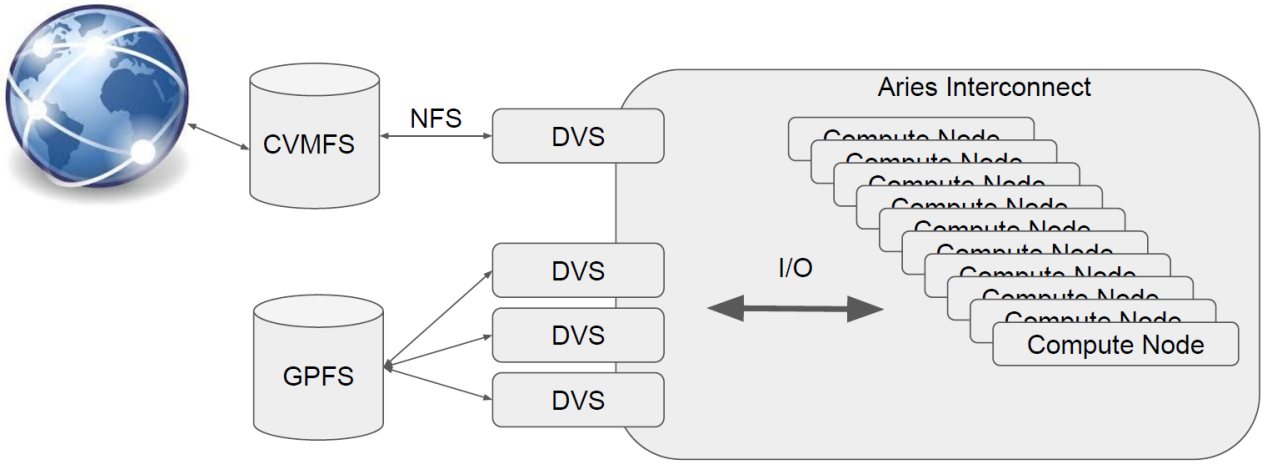


Cori – NERSC Cray XC-40 Supercomputer

- 20 TB $SCRATCH/user (Luster FS)
- 2,004 Xeon "Haswell" nodes
  - 32 Cores (64 vCores)
  - 120 GB RAM (~ 1.8 GB / vcore, plenty for STAR)
- 9300 Xeon Phi "Knights Landing" nodes (KNL)
  - 68 Cores (272 vCores)
  - 96 GB RAM (0.35 GB / vcore or 1.4 GB / core)

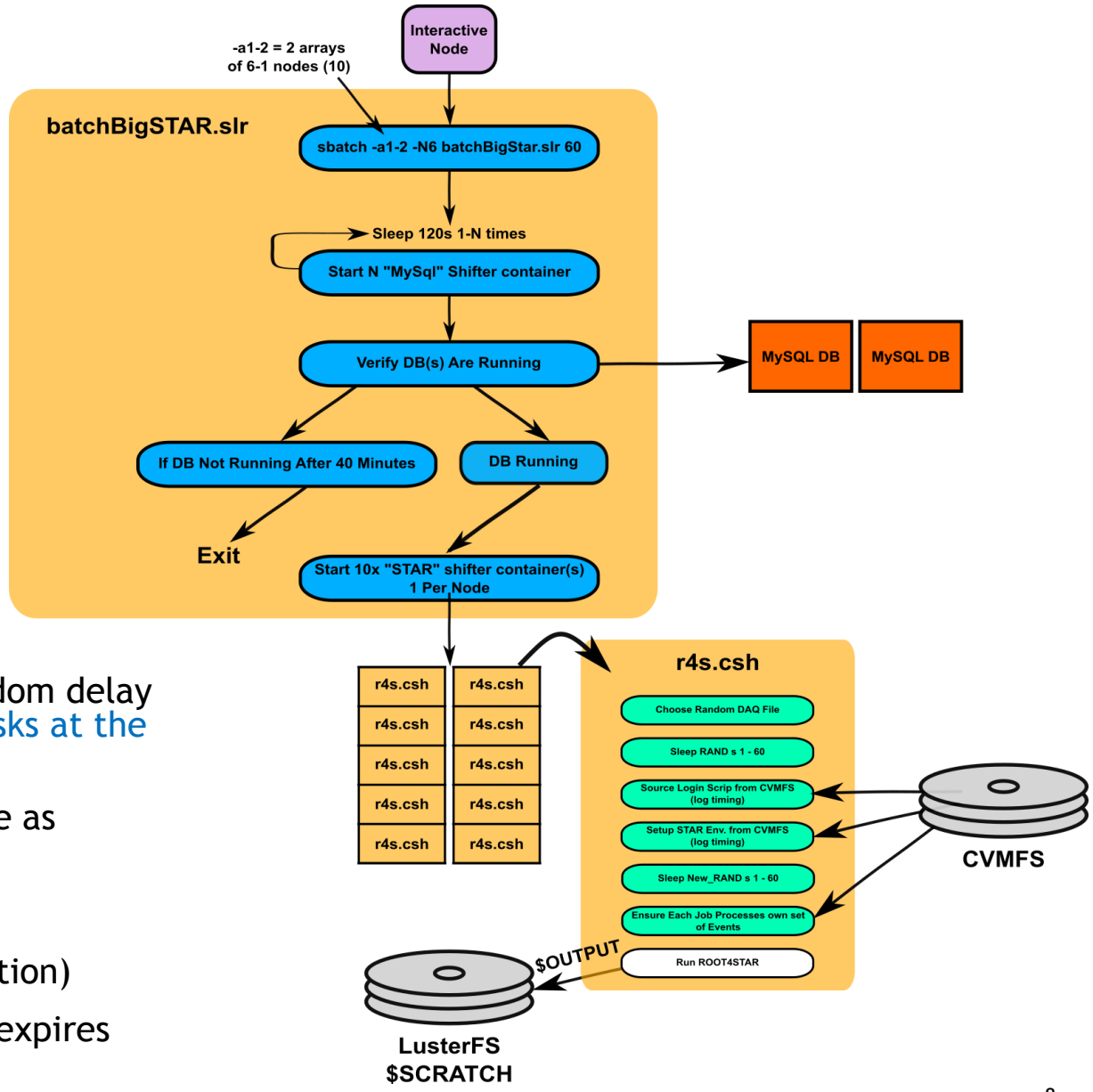
- Fuse restriction on Cori = cannot have CVMFS directly
- DVS (Cray Data Virtualization Service) system does I/O Forwarding and data caching
- Cori has 32 DVS Servers, 4 dedicated to CVMFS

Very different core/memory configs, DVS layers, … Best approach: scale up # of tasks, look at # of events or files produced per unit of time as the criteria for reaching optimal throughput

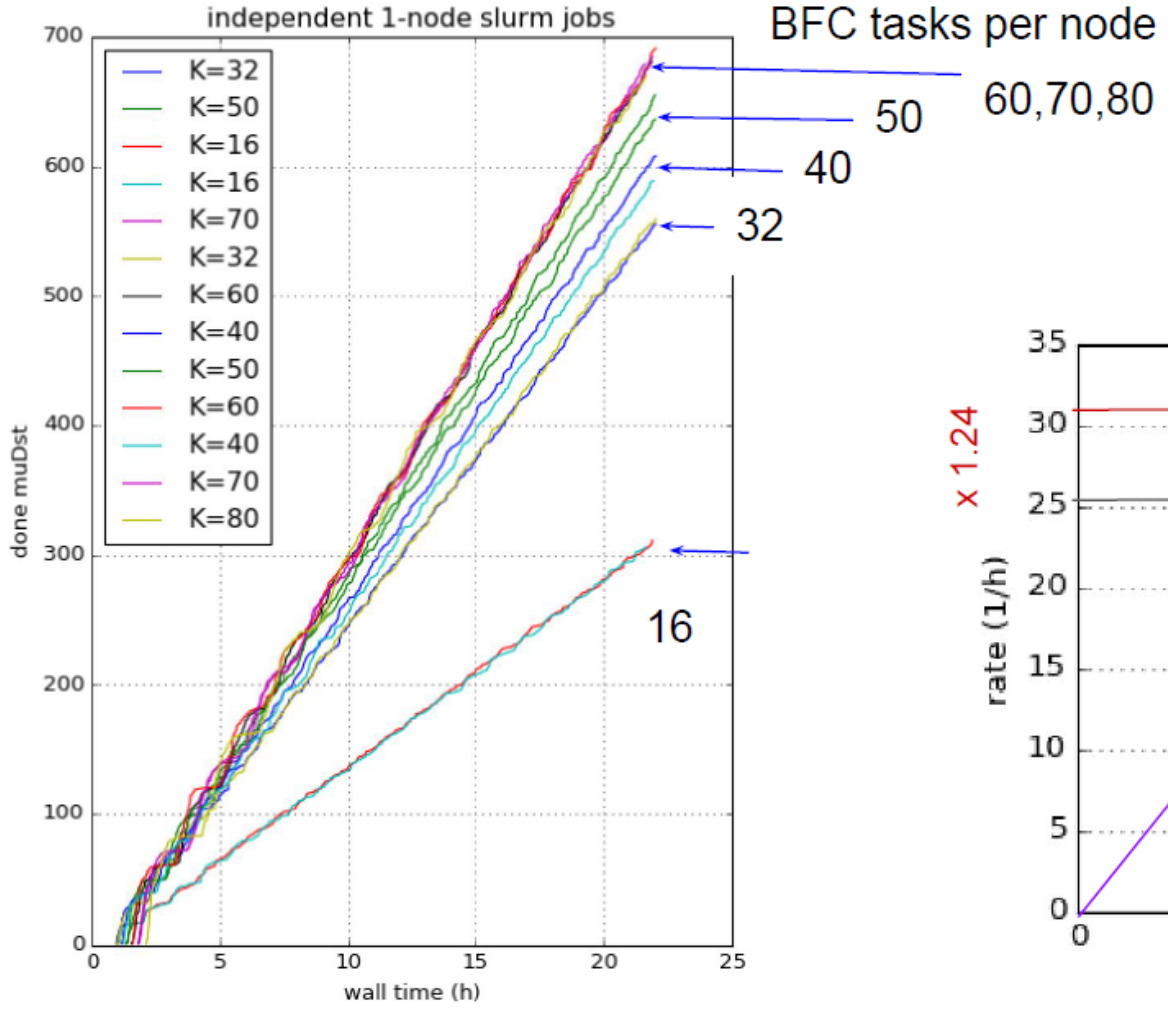


STAR Workflow on Cori

- Batch submission starts on interactive node – all tasks could start at the same time
- `batchBigStar.slr` fires "MySQL" shifter container, verifies service starts, then runs STAR's task steering script `r4s.csh`, one per batch node
  - 1 DB server per 5 nodes – we did not want to test MySQL service scalability (would be OK up to 1 for 10-15)
  - Sleep intervals between each DB copy to spread load (do NOT want the DB to be hit by jobs at the same time)
- `r4s.csh` script
  - Source the STAR and code version environment once. A random delay is inserted before the sourcing – avoids sourcing from all tasks at the same time (remember! Scale are 50 to 100k tasks)
  - Our script runs multiple "tasks" – we used 60 tasks per node as "sweet spot" – again, small delay inserted
- We used 1 input file per node
  - each task processes part of the input file (event parallelization)
  - Best processing model – run over files until time allocation expires
- All output written to Luster $SCRATCH

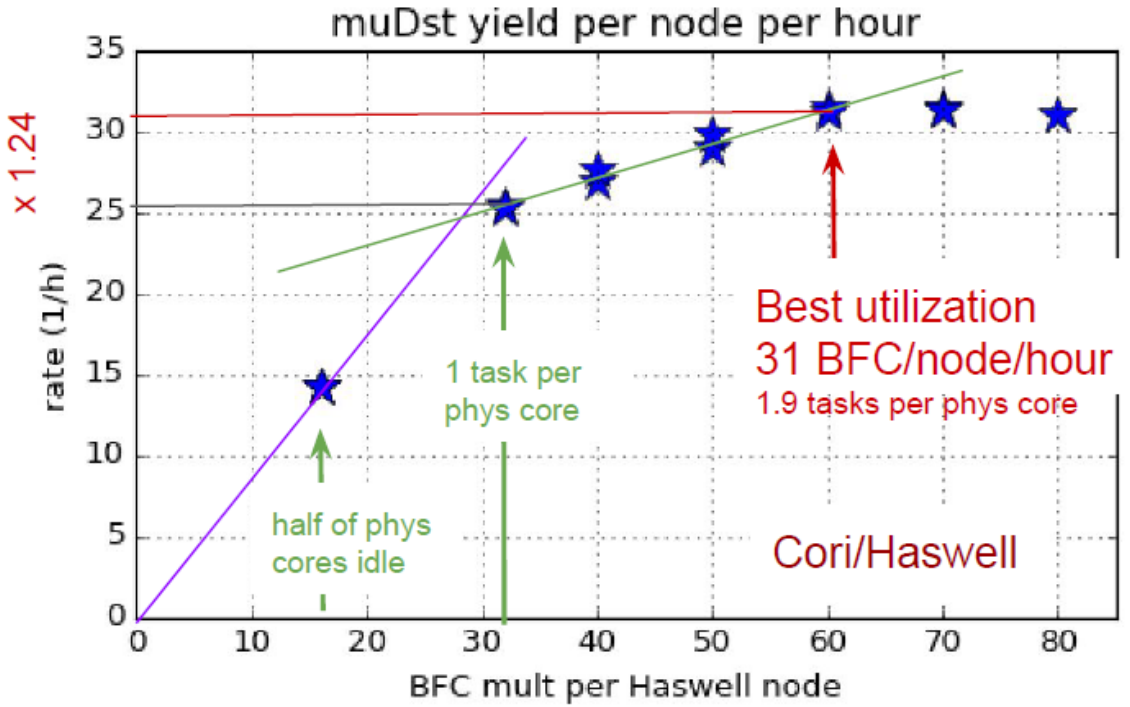


STAR Event Rate Vs. Task Density (Haswell)



Haswell: 32 phys cores,
64 vCores, 125 GB RAM

Code; janNersc/starExp/Bfc-effi-multiThreads
PATH_DAQ=/project/projectdirs/mpccc/balewski/star_daq_2016
STAR_VER=SL17d
BFC_String="DbV20161216,P2016a,StiCA,mtd,mtdCalib,btof,PxlHit,IstHit,SstHit,beamline3D,picoWrite,PicoVtxVpd,BEmcChkStat,-evout,CorrX,OSpaceZ2,OGridLeak3D,-hitfilt"



Balewski, J., Porter, J., Rath, G., Lee, R., Quan, T. (2018) PDSF – Status & Migration to Cori *HEPiX Fall 2018, Barcelona*

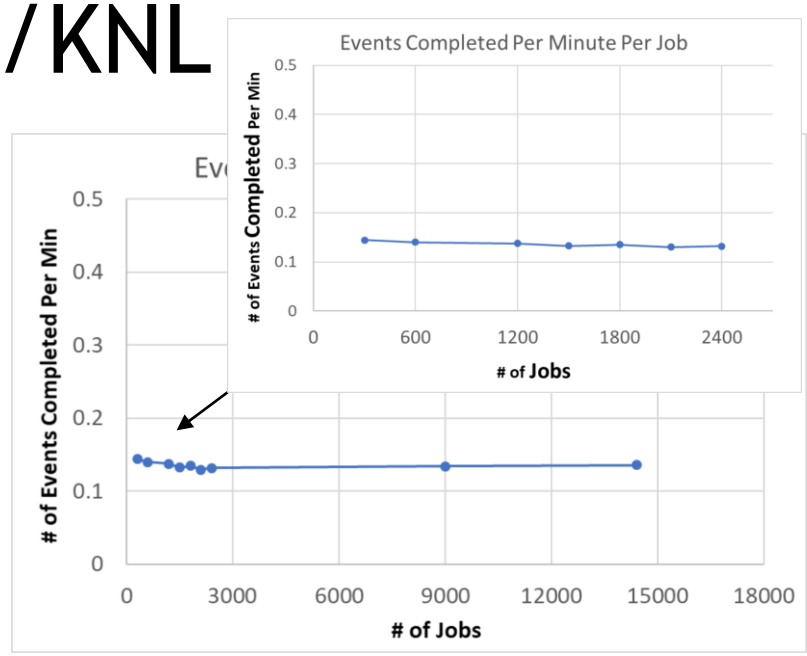
Testing CVMFS usage @ Cori/KNL

Reservation

- **1 (50 KNL node)** - Small steps testing at first …
- **2 (300 KNL node) -** 2400, 9000, 14400 jobs batches submitted

How does it look like in terms of throughput maximization?

- Looked at average of events per min per "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



Goal was to see how CVMFS scales over NFS at Cori when serving STAR Software

- Workflow is: (a) load STAR environment / login script (b) setup env for a given version of the library (c) run tasks
- Logged timing to source login script (a) & setup STAR environment (b) both loaded from CVMFS
- Curve stays flat as the # of nodes increase
  - At large umber of tasks, error bar gets big – there is an IO pile-up but acceptable
  - Within our expected scalability range in terms of tasks / nodes, CVMFS setup at NERSC scales well!

Observations & Recommendations, …

- **Installing the full STAR Software inside Docker containers is self-sufficient but not a practical solution.**
  - Use of CVMFS for software provisioning lowers the bar of maintenance (only one container per OS).
- **Testing CVMFS scalability needs to be thought out**
  - For running on resources you control (like our online cluster), know your software stack size well - make sure you have sufficient cache (plan for the future) – in our cases, 3 GB (< 1 GB of binary) -> 25 GB cache
  - Local batch setup would start jobs one by one as slots appear over a large period of time (there is a natural spread) – Once loaded, the same tasks or jobs would re-use what is in cache (as intended)
  - On Cori, with advanced reservations, the situation is NOT comparable to a standard batch setup: Several 10k jobs may start simultaneously – introducing a flexible random delay is needed before each step – this spreads the load and avoids clogs
    - Layers such as DVS caches data, NOT MetaData – avoid MD lookups (ls –l, find, …) – those will force a hit back CVMFS servers
- **Consider your workflow as a three step process: (a) load the experimental environment (b) setup the "version" of the code to be run (c) run a task [or job]**
  - Everything needs to be scripted, timed, and printed to the screen to troubleshoot – it will help evaluating scalability as you go from 1, x10, x100, x1000, x10k, x100k in number of tasks
- **Find the optimal number of tasks to be run**
  - Each HPC system has its own core/memory setup – you need to find an optimal number of tasks per node. Study it (more than a simple memory/core issue). Evaluate (if you need a DB), your many tasks per DB is sustainable – plan accordingly and create task dependencies

Conclusion

- We followed all the steps shown …
  - With delays randomly spread over 60 seconds for loading the environment (and starting tasks), our tests showed jobs scaled well up to 15k jobs
  - Timing for loading the environment was rather flat (wide spread at higher values may indicate delays due to environment loading concurrency)
  - Our job throughput remained stable over the full range
- In practice, what was found in STAR was also used for “A”nother experiment
- We had run before (fully loaded Container) @ 26k cores (we have run real-data reconstruction before – we are also running “embedding” now)
- Our next steps
  - Resume processing at full scale with our current approach!
  - More studies needed to find the best workflow for maximal use of time allocation (“convoy” model)
  - Possibly evaluate commonalities across experiments (a 3 stages workflow could be generalized)
  - More advanced: Find ways to auto-tune the number of tasks per node??