Experience with containers on Titan supercomputer at OLCF

Sergey Panitkin
(BNL)

Outline

- Introduction
- Singularity on Titan
- Container build setup
- Container tests on Titan
  - Performance of ATLAS simulations in containers
    - Singularity “tune”
  - I/O Properties
- Summary

Containers in ATLAS

- Containers are viewed as a light weight virtualization technology that allows to run experiment/user specific environment regardless of host site environment
- ATLAS started testing containers on Grid in 2017
  - Docker, Singularity
    - Typically requires Centos 7 installed on a site for full Singularity support
    - Site Singularity configuration plays large role
- Containers for HPC were tested at NERSC with Shifter and Singularity
- For HPC machines without CVMFS (like Titan) containers are viewed as a software distribution tool

Containers on Titan

- Singularity container platform became available for tests on Titan in 2017
  - Accessible on batch worker nodes and interactive worker nodes
  - Supported semi-officially
  - Some documentation and scripts are available in github
- Currently Singularity v2.4.0 is installed as a module
- Singularity on Titan imposes a few requirements on user container images
  - No run-time mount points, all file system bindings have to defined in the image. Run time bindings (-b fs1:fs2) are not supported, since CNL kernel does not support overlayfs. (Singularity on Summitdev@OLCF machine supports this option)
  - A placeholder file for Titan specific setup script in the image (to be invoked at run time)

Container build for Titan I

- Singularity installed from scratch on my laptop, since root privileges are needed for container image building
  - MacBook Pro 2016 laptop with VirtualBox, Vagrant VM with Singularity 2.4, following Singularity documentation
  - Manual install of Singularity v2.4.2 in Vagrant VM later on. A lot of bug fixes in this version.
- Build Singularity images with CentOS 6, 7 as base OS, imported at build time from Docker Hub
  - Tried several different OS versions, did not see much difference for container performance
  - Added a few system libraries required by ATLAS software
  - Some extra rpms for common tools required for ATLAS release install scripts (git, perl, wget, …)
- “Post”-stage script (from Adam Simpson’s (ORNL) Github) was used to define Titan specific mount points
  - Makes Titan’s shared file system visible at run time
- ATLAS release 21.0.15 installed using Pavlo’s scripts from Github
  - Same script is used for ATLAS releases installation on Titan
  - Current production release for Geant simulations on Titan
- ATLAS DBRelease installed
  - Special handling for installation of ATLAS DBRelease fix for rel. 21.0.15
  - Installed DBRelease configuration files customized for the container
- Several users added with proper Titan userIDs – required for asetup

Container build for Titan II

- Image build time ~2 hours on MacBook Pro
  - Max system load during build ~40%
- Container file sizes
  - Image file on top of Ext3 filesystem ~29GB
  - SquashFS based image file ~7GB
    - Support for SquashFS was introduced recently in Singularity
    - SquashFS supports compression
  - Same ATLAS release installed directly on Titan's FS: ~27GB
- For comparison: some containers build at BNL by Wei Yang (SLAC)
  - "Fat" ATLAS container ~600GB
    - Full ATLAS (deduplicated) CVMFS tree
  - Container with rel. 21.0.15 and DBRelease ~ 50GB
    - Also extracted from CVMFS

ATLAS container tests on Titan

- Ext3 and Squash containers were copied to Lustre and NFS on Titan
    - Tried several container placement options including RAMdisk
- Tested with an ATLAS production job
    - Short, single node job with 16 events on 15 CPU cores
- Jobs submitted manually to the batch queue, f.e.
    - aprun -n 1 -N 1 -d 15 -r1 singularity exec /ccs/proj/csc108/AtlasReleases/containers/my_centos_6_docker_Titan_DBRelease_with_gcc_v2.simg ./run.sh
    - Release setup done at run time via shell script (run.sh)
    - Job working directory is on Lustre or RAMDisk depending on the test
    - Root input file with events on NFS, Lustre or RAM disk depending on the test
- Timing information from Athena logs

ATLAS container tests on Titan: First results

Type	Location	Size, GB	Setup time, s	Run time, s	Job ID
Direct Release	NFS	26.7	357	1610	3801346
SquashFS	NFS	7.2	742	4272	3800895
Ext3	NFS	29	766	4029	3801075
SquashFS	Lustre	7.2	746	4157	3807410
Ext3	Lustre	29	773	4023	3807409
SquashFS	RAM disk	7.2	722	4124	3801346

Setup time: from the transformation start to the event loop start
Run time: from the transformation start to exit

Some initial observations:

- Simulations in containers run ~x2 longer than the simulation ran from disk installed release
  - Unexpected!
  - Is this related to access to the large container files?
- No big difference between run times for containers placed on NFS or Lustre (NFS is optimized for read and is used for software installation on Titan)
- No big difference between Ext3 and SquashFS based containers
- Container started from RAM disk on worker node runs similar to the containers on shared FS
  - Indication that the slowdown is not IO related?!

Solving the slow containers puzzle

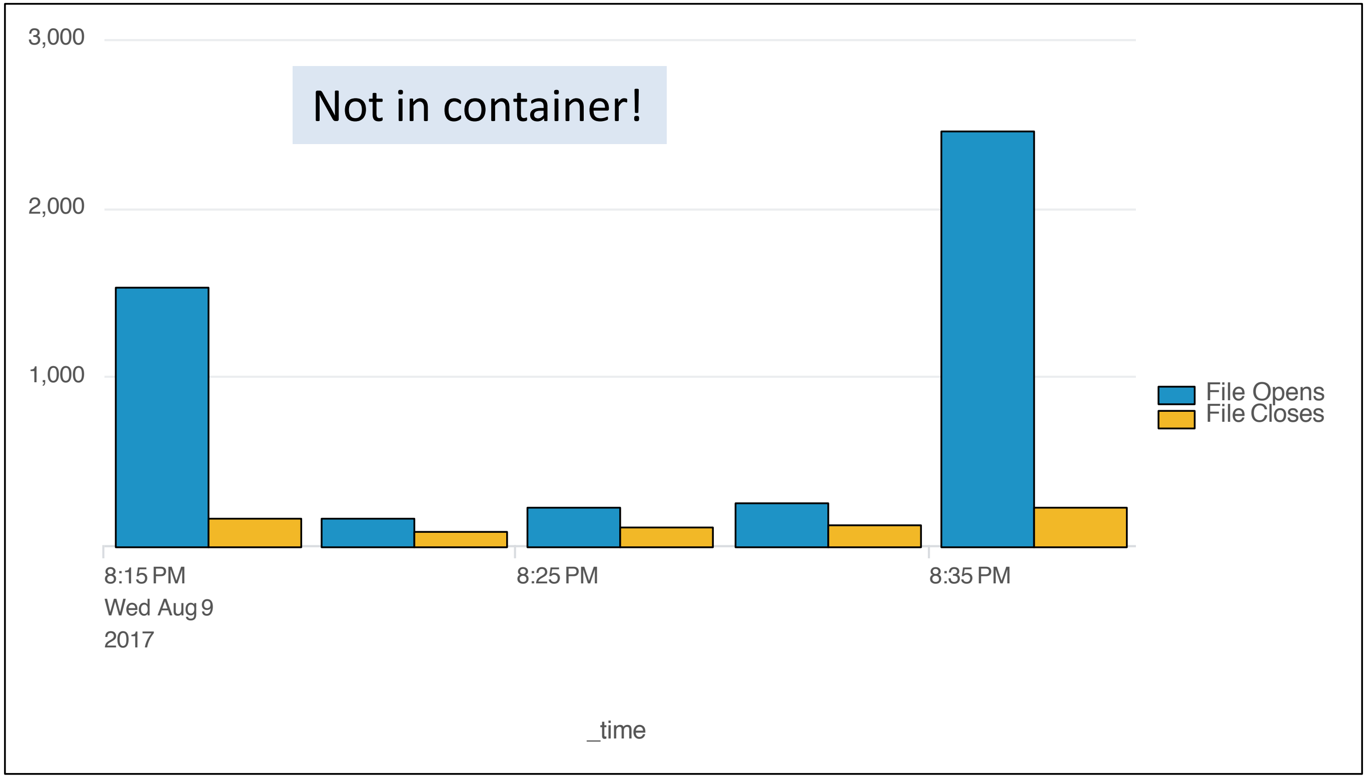
- After discussing test results with Adam Simpson (OLCF) and trying several other possibilities I looked at ld-intercept feature
- This feature is active by default on Titan and is used to intercept loading of MPI related shared libraries within containers
- Probably an important feature for other use cases but not needed for ATLAS simulations
- In the current Singularity setup at OLCF dl-intercept is always "ON" but can be switched off after Singularity module is loaded with:
  - unset SINGULARITYENV_LD_AUDIT

Running with DL AUDIT OFF

Type	Location	Size, GB	Setup time, s	Run time, s	Job ID
Direct Release	NFS	26.7	357	1610	3801346
SquashFS ld_audit ON	NFS	7.2	742	4272	3800895
SquashFS ld_audit OFF	NFS	7.2	221	1425	3822559
Ext3 ld_audit OFF	NFS	29	239	1491	3822317

- Simulations in containers run ~x3 faster when LD AUDIT is turned off. Good!
  - "unset SINGULARITYENV_LD_AUDIT" works!
- Simulations in containers now run noticeably faster than in case with ATLAS release installed on NFS. Good!
  - ~1.5 min. improvement in transformation start up time
  - ~3 min. improvement in overall run time
- Not much difference in performance between SquashFS and Ext3 based containers
  - No visible penalty for using compression in SquashFS . Good!
    - Perhaps SquashFS container is even a bit faster
  - SquashFS based containers are much smaller (x4). Good!
- Significant improvements in IO in case of container (see next slides). Very good!
  - Much lower load on Lustre metadata server due to change in file access pattern
    - Single file access for Singularity container vs multiple files access for release installed on disk (direct release)
    - NB: ATLAS simulation reads/loads hundreds of files (Python scripts, shared libraries, etc) during execution especially at start up

Typical Splunk profile for a single AthenaMP job on Titan



Not in container!

- The figure shows file open/close operations on Lustre MDS as a function of time
- Single AthenaMP production test job with 16 workers, 16 events
  - Atlas release on NFS. Job working directory on Lustre. sqlite200 DB in RAM
- Large spike in reads at Athena start up
- Large spike at the end of the job due to merging of AthenaMP workers outputs
- More file open() than file close() operations!
  - Shared libraries and Python includes searches. Confirmed with strace profiling

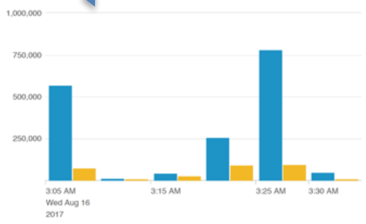
Evolution of IO profiles for ATLAS production jobs on Titan

350 jobs running in parallel. Not in container

Problem!

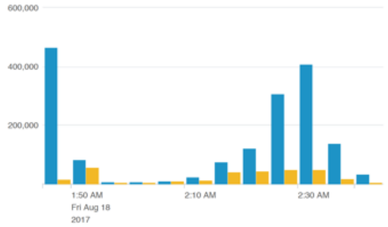
Much better!

Before optimization



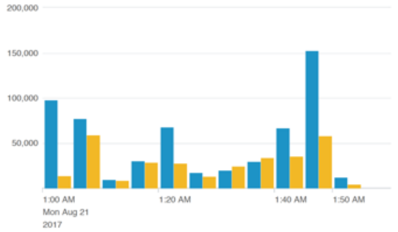
Start Time: Aug 15, 2017 23:56:11
End Time: Aug 16, 2017 01:01:16
Nodes Used: 350
batch id: 3567126

Environment cleanup



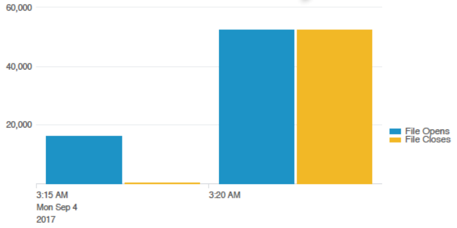
Start Time: Aug 18, 2017 01:46:04
End Time: Aug 18, 2017 02:58:22
Nodes Used: 350
batch id: 3570319

ATHENA 'hack'



Start Time: Aug 21, 2017 01:02:42
End Time: Aug 21, 2017 02:16:04
Nodes Used: 350
batch id: 3573737

Working dir in RAM Disk



Start Time: Sep 04, 2017 03:19:01
End Time: Sep 04, 2017 04:00:00
Nodes Used: 350
batch id 3602038

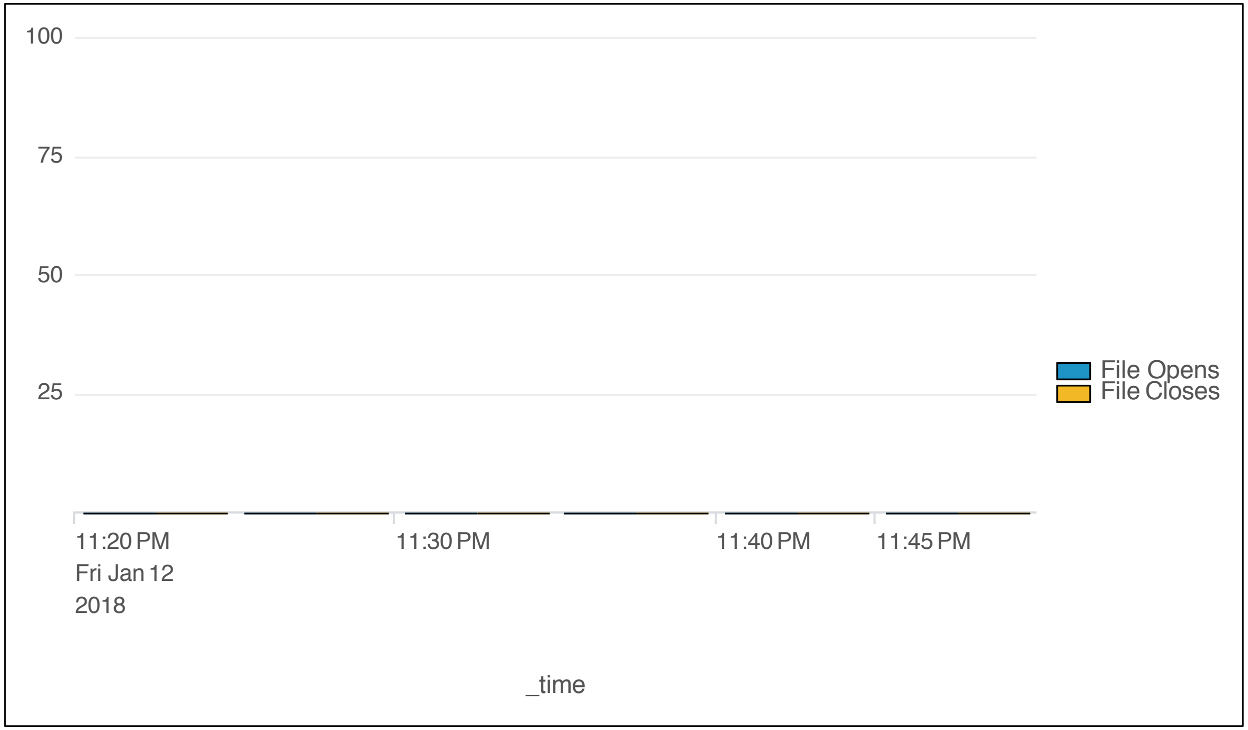
- Significant reduction of IO. Number of 'open' operations almost matches with number of close operations.
  - Initial spike came form MPI wrapper which used to launch ATHENA Job on computing node. Already reduced with same fix like athena.py
- Current setup of ATLAS production at OLCF
  - ATLAS releases: NFS
  - Job working directories and input data: RAM disk of computing node
    - Output data moved to Lustre at the end of the job

Ack: Danila Oleynik

We can run simultaneously up to 20 of such job groups. I/O can limit scalability

Container I/O. I

- Splunk profile for simulation in SquashFS based container located on NFS
- The plot shows Lustre file open()/close() operations
- Almost no file open/close on Lustre!



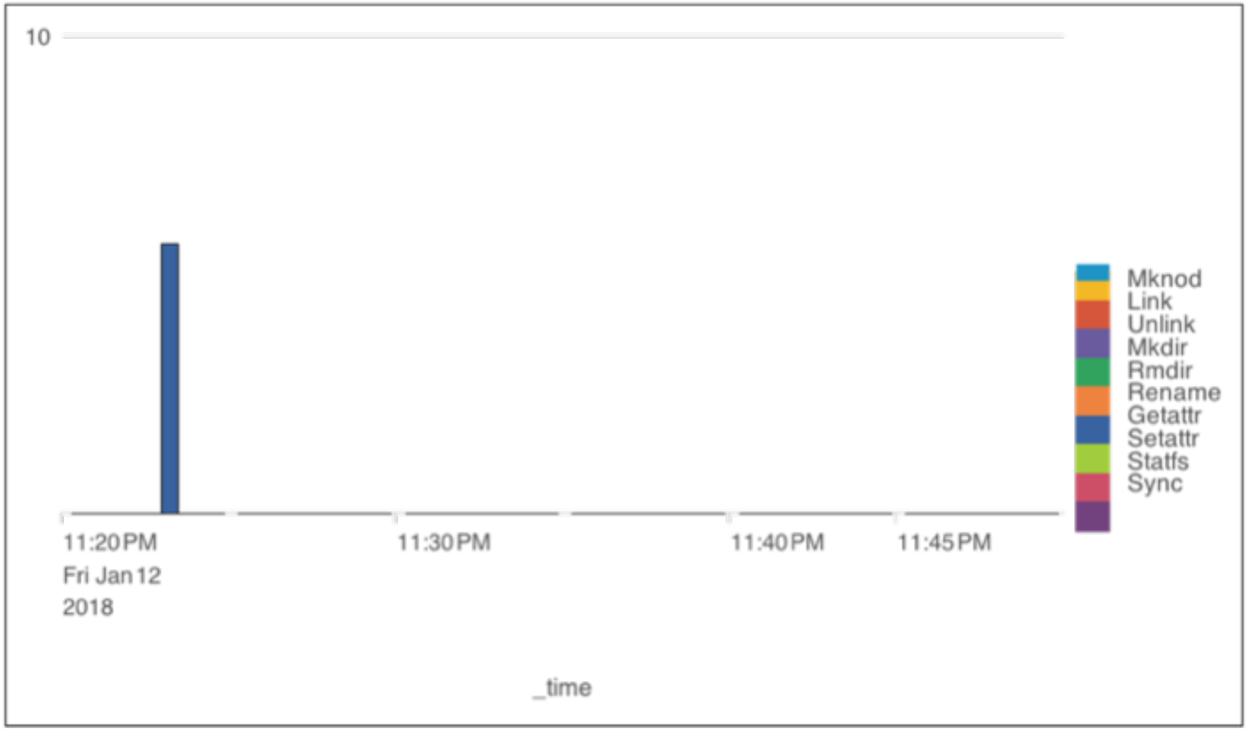
Job 3822559

To compare with previous slides!

Container I/O. II

- Splunk profile for simulation in SquashFS container located on NFS
- Very low metadata activity on Lustre shared filesystem

Job Specific I/O Statistics: Other Metadata Operations



Job 3822559

Containers on RAMdisk

Type	Location	Size, GB	Setup time, s	Run time, s	Job ID
Direct Release	NFS	26.7	357	1610	3801346
SquashFS ld_audit ON	NFS	7.2	742	4272	3800895
SquashFS ld_audit OFF	NFS	7.2	221	1425	3822559
SquashFS ld_audit OFF	RAMdisk	7.2	209	1477	3828894
SquashFS ld_audit OFF	RAMdisk+	7.2	208	1481	3828925

RAMdisk+ : container, input data and working directory on RAMdisk

- Small size SquashFS containers allow placement on RAMdisk on Titan
  - Container copy time ~40s
- Tests show that job start up time for RAMdisk based container is ~2.5 min shorter compared to running with disk based (direct) release
- Tests show no significant acceleration in setup or run time compared to container on NFS
  - Timing changes consistent within normal runtime fluctuations on Titan
  - Probably already reached IO performance plateau. Effective cashing at FS level.

Summary

- Started work with Singularity containers for Titan
- ATLAS simulations in containers performed well (after the default Singularity option is turned off)
  - Start up and run time improvements compared to standard release install on shared file system
- Containers showed very good IO properties with almost no load on Lustre MDS
  - Due to change in file access pattern
  - Important from operational point of view for the host site
  - Important for scalability
- Use of containers should allow to scale up number of simultaneously launched jobs, especially with Harvester in production on Titan

Plans

- Jan.- May. Containers on Titan for ATLAS
  - Scaling studies.
    - MPI wrapper for containers
    - Study strong scaling
  - Work with Danila on using containers in ATLAS production on Titan
    - Integration with current Pilot setup (with Danila)
    - IO properties and timing for production
  - Containers integration with Harvester (with Danila and Pavlo)
    - Pavlo launched my container on Titan via Harvester this week
  - Containers created by ATLAS (Wei Yang) at BNL
    - Need to be build and configured to reflect Titan specifics
    - Large size, long transfer times, hard to modify on a laptop
    - Started work with these, still do not work 100% on Titan, transformation crashes
      - Titan specifics, DB configuration
    - Hope to converge on a working container
  - Work on container build machine at BNL
    - Discussions yesterday with Doug , Wei and Xin Exchanged ideas on creation of automated container build and distribution system for US HPC
- Containers with NGE (with Matteo Turilli)
    - Containerized ATLAS simulations are probably the easiest case for NGE tests