Containers on Titan

Sergey Panitkin
(BNL)

Introduction

- ATLAS started testing containers on the Grid in 2017
  - Two run times depending on sites preferences: Docker and Singularity
  - Typically requires Centos 7 installed on a site for full Singularity support
  - Site Singularity configuration plays large role
- Containers are viewed as software distribution tool for HPC machines without CVMFS
- Containers for HPC were tested at NERSC with Shifter and Singularity
- Shifter containers are in production at NERSC

Containers on Titan

- Singularity container platform became available for tests on Titan in 2017
- Accessible on batch worker nodes and interactive worker nodes
- Currently v2.4.0 module is available
- Some documentation and scripts are available in github
- Singularity on Titan imposes several 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 supports this option)
  – Placeholder for Titan specific setup script in the image (to be invoked at run time)
  – Linux userIDs in the image should coincide with Titan's userIDs

Container build for Titan

- Singularity images build from scratch
- Images with CentOS 7 as base OS loaded at build time from Docker Hub
  - Added a few system libraries required by ATLAS software
- “Post”-stage script for Titan specific mount points (from Adam Simpson’s Github)
- ATLAS release 21.0.15 installed using Pavlo’s scripts
  - Special handling for installation of ATLAS DBRelease fix for 21.0.15
  - Installed customized DBRelease configuration files for the container
  - Some extra rpms for common tools required for ATLAS release install scripts (git, perl, wget,…)
  - Strace for IO profiling
- Several users added with proper Titan userIDs to allow asetup to run
- SquashFS based image ~7GB installed on Titan’s NFS (~x4 smaller than other image formats due to compression)

Initial ATLAS container tests on Titan

- Ext3 and Squash containers were copied to Lustre and NFS on Titan
- Tested with ATLAS production job
  - Short jobs with 16 events
- Jobs submitted manually to 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 run.sh
  - Job working directory is on Lustre
  - Root Input file with events on NFS or Lustre
- Timing from Athena logs
- Tried several container placement options including RAMdisk

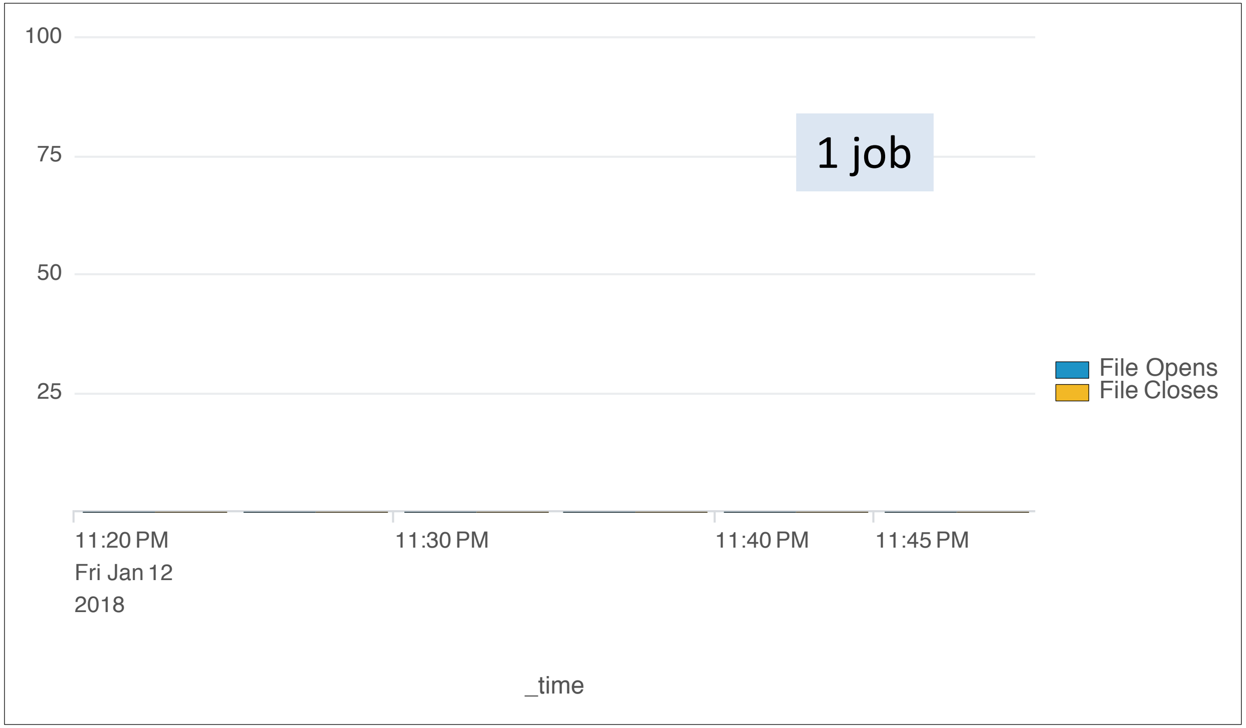
Running with DL AUDIT OFF

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

Container I/O

- Splunk profile for simulation in SquashFS container located on NFS
- Lustre file opens/closes



Job 3822559

Containers with MPI wrapper

- Scalability tests with MPI wrapper for production like setup

Call sequence

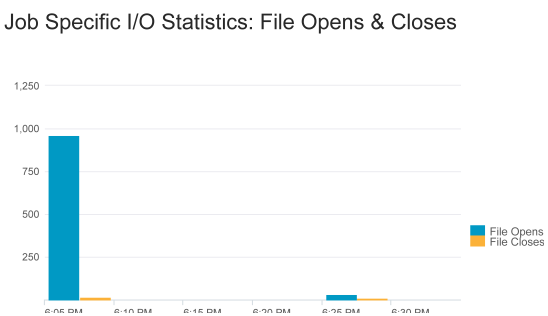
MPI wrapper → Setup script → container → Run script

- Run script runs asetup and Sim_tf.py transformation
- Use RAM Disk for worker directory and transient files
- Splunk profile and strace tracing for IO monitoring

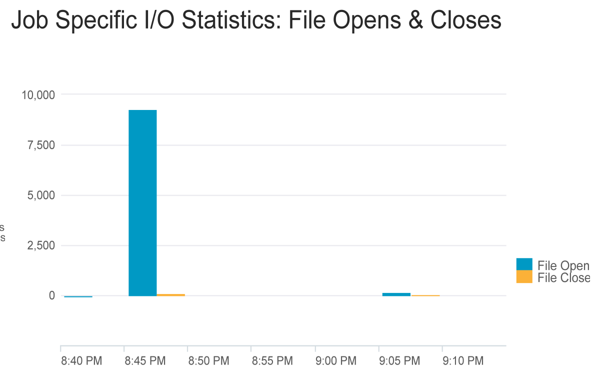
Container scalability test

MPI wrapped container with ATLAS detector simulation

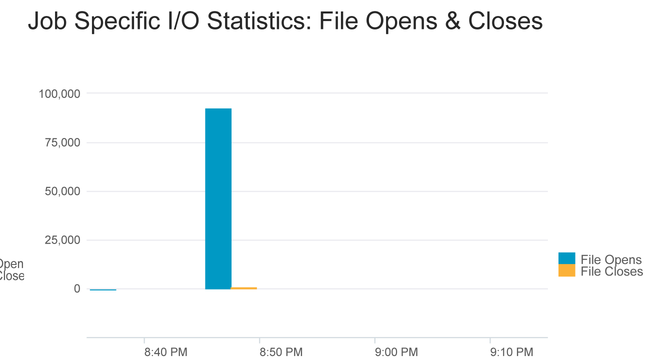
1 rank



10 ranks



100 ranks



- ~900 open() operations at job startup for a single rank
- Linear growth in number of opens with increase in number of ranks
  - Single rank properties dominate the I/O
- Where the IO load comes from?
- Since Splunk does not provide details use strace to look at IO details
- Use approaches developed during the Summer 2017 “IO crisis”
  - RAM disk for working directory, Environment clean up, etc

Monolithic MPI wrapper

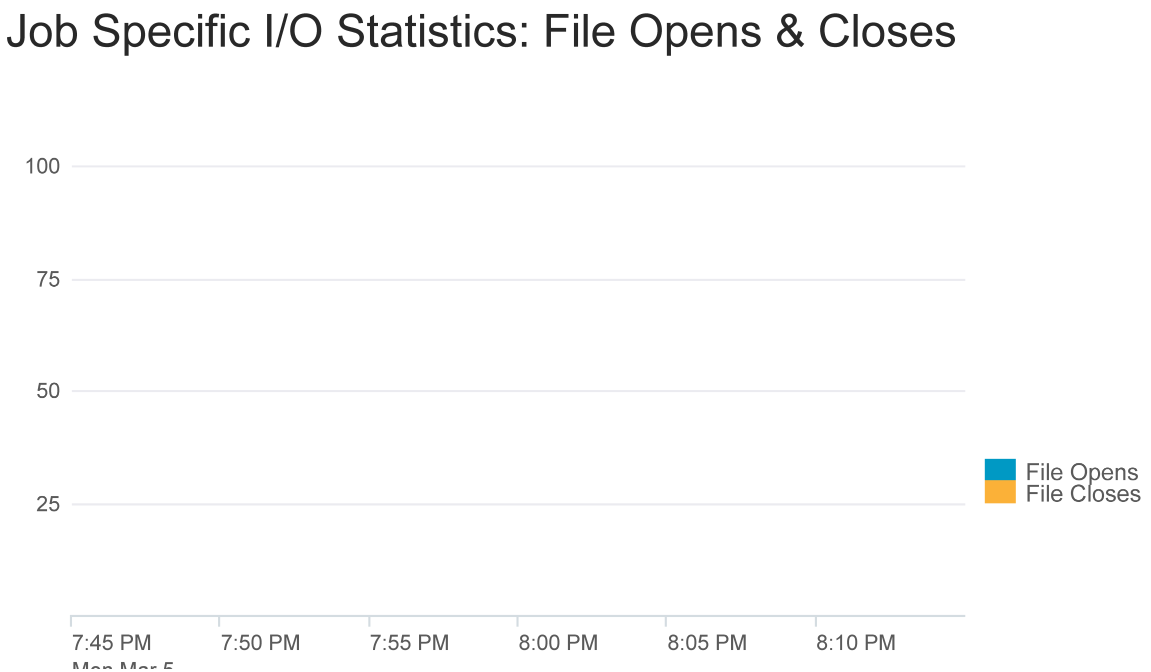
- MPI wrapper is a python script
  - Uses several Python modules
- Modules are loaded via search in $PYTHONPATH – a procedure known to generate calls to Lustre on Titan
- One can build a monolithic executable with all modules pre-included so that no $PYTHONPATH search is done
- One of the available tools: pyinstaller
- pyinstaller --onefile multi-job_container_v1.py
- Used this to run containers

Component analysis example

Monolithic MPI wrapper, Fixed Python environment, RAM disk

- **Strace of the mpi wrapper: No calls to Lustre**
- Job 3913979
- panitkin@titan-ext2:/lustre/atlas2/csc108/proj-shared/panitkin/containers_tests/scalability_tests/test_2_strace_small_wrapper_synlegacy_5> cat trace_mpi_wrapper.out | grep -i "open(" | grep -i lustre | wc -l
- 0
-
- --**Strace of singularity: No calls to Lustre**
- Job 3914142
- panitkin@titan-ext2:/lustre/atlas2/csc108/proj-shared/panitkin/containers_tests/scalability_tests/test_2_strace_small_wrapper_synlegacy_5> cat trace_singularity.out | grep -i open | wc -l
- 16
- panitkin@titan-ext2:/lustre/atlas2/csc108/proj-shared/panitkin/containers_tests/scalability_tests/test_2_strace_small_wrapper_synlegacy_5> cat trace_singularity.out | grep -i open | grep -i lustre | wc -l
- 0
- **Non Luster calls**
- panitkin@titan-ext2:/lustre/atlas2/csc108/proj-shared/panitkin/containers_tests/scalability_tests/test_2_strace_small_wrapper_synlegacy_5> cat trace_singularity.out | grep -i open
- 17:51:18 open("/etc/ld.so.preload", O_RDONLY) = 3
- 17:51:18 open("/tmp/scratch/tmp/_MEIHyes91/tls/x86_64/libreadline.so.5", O_RDONLY) = -1 ENOENT (No such file or directory)
- 17:51:18 open("/tmp/scratch/tmp/_MEIHyes91/tls/libreadline.so.5", O_RDONLY) = -1 ENOENT (No such file or directory)
- 17:51:18 open("/tmp/scratch/tmp/_MEIHyes91/x86_64/libreadline.so.5", O_RDONLY) = -1 ENOENT (No such file or directory)
- 17:51:18 open("/tmp/scratch/tmp/_MEIHyes91/libreadline.so.5", O_RDONLY) = 3
- 17:51:18 open("/tmp/scratch/tmp/_MEIHyes91/libdl.so.2", O_RDONLY) = -1 ENOENT (No such file or directory)
- 17:51:18 open("/lib64/tls/x86_64/libdl.so.2", O_RDONLY) = -1 ENOENT (No such file or directory)
- 17:51:18 open("/lib64/tls/libdl.so.2", O_RDONLY) = -1 ENOENT (No such file or directory)
- 17:51:18 open("/lib64/x86_64/libdl.so.2", O_RDONLY) = -1 ENOENT (No such file or directory)
- 17:51:18 open("/lib64/libdl.so.2", O_RDONLY) = 3
- 17:51:18 open("/tmp/scratch/tmp/_MEIHyes91/libc.so.6", O_RDONLY) = -1 ENOENT (No such file or directory)
- 17:51:18 open("/lib64/libc.so.6", O_RDONLY) = 3
- 17:51:18 open("/tmp/scratch/tmp/_MEIHyes91/libncurses.so.5", O_RDONLY) = 3
- 17:51:18 open("/dev/tty", O_RDWR|O_NONBLOCK) = -1 ENXIO (No such device or address)
- 17:51:18 open("/proc/meminfo", O_RDONLY) = 3
- 17:51:18 open("/sw/xk6/bin/singularity", O_RDONLY) = 3

Splunk profile for a fixed MPI job



Summary

- ATLAS simulations in containers performed well (after the default Singularity option is turned off)
- Containers showed good IO properties when running ATLAS simulations with almost no load on Lustre MDS
- Containers scale quite well due to encapsulation of I/O in container file that is placed on read-optimized file system as well as usage of RAM disk on worker nodes.
- Residual IO imprint on Luster is due to Python based MPI wrapper environment and that can be controlled via environment clean up
- Started working on integration with ATLAS Harvester instance at OLCF

Plans

- Work with Danila on using containers in ATLAS production on Titan (Apr-May, already started , will work on it this week)
  - Containers integration with Harvester (with Danila and Pavlo)
- Work on containers created by ATLAS. Ongoing discussion within ATLAS
    - Need to be built and configured according to ATLAS standards/requirements but also to reflect Titan specifics
    - Common issue for HPC sites (especially in US)
    - Hope to converge on working scheme sometimes in May
- Containers with NGE (with Matteo, when he’s available)
    - ATLAS simulations are probably the easiest case

Integration with ATLAS Container project

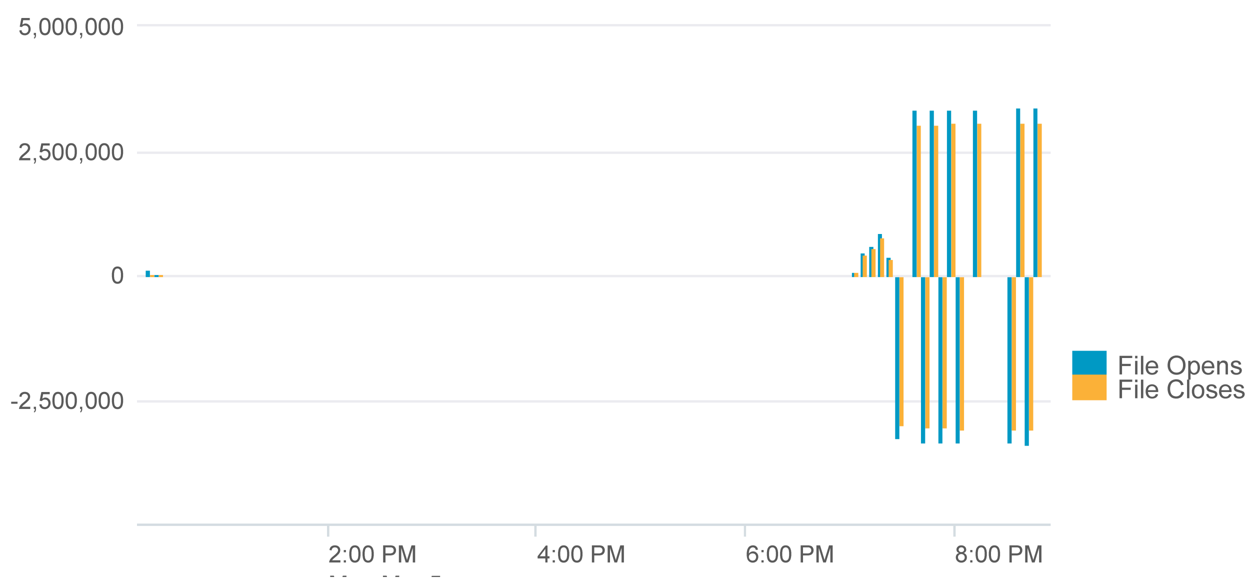
Discussion D. Benjamin, A. Forti, P. Love, A. De Salvo, T.Childers, W.Yang, X. Zhao,….

- Some of the issues that are being discussed
  1) container naming convention
  2) single release, multiple releases, all releases
  3) conditions db or not
  4) how can PanDA/Harvester know what container to use when running on HPC
  5) how do we handle site customizations
  6) container creation system
  7) container distribution
  8) how does Harvester know where to find a container on an HPC
- ATLAS meeting next week (?)

Back up slides

Splunk artefacts

Job Specific I/O Statistics: File Opens & Closes



Negative entries in Splunk. Data corruption?