Challenges in Delivering and Deploying Software at Scale in Large Clusters

Douglas Thain and Kyle Sweeney
University of Notre Dame

{dthain|ksweene3}@nd.edu
Software Deployment on HPC

• Classic Approach
  – Single process MPI app created by end user.
  – Sysadmin installs, tests, proves the application.
  – Adjust to exploit local libraries / capabilities.
  – Application satisfied with a single site.

• Evolving Approach
  – Complex stacks of commodity software.
  – Developer is not the user!
  – Installed by end user just in time.
  – Users migrate quickly between sites.
Problem: Software Deployment

- Getting software installed on a new site is a big pain! The user (probably) knows the top level package, but doesn't know:
  - How they set up the package (sometime last year)
  - Dependencies of the top-level package.
  - Which packages are system default vs optional
  - How to import the package into their environment via PATH, LD_LIBRARY_PATH, etc.

- Many scientific codes are not distributed via rpm, yum, pkg, etc. (and user isn't root)
"I just need BLAST."
"Oh wait, I need Python!"
"Sorry, Python 2.7.12"
"Python requires SSL?"
"What on earth is pcre?"
"I give up!"
VC3: Virtual Clusters for Community Computation

Douglas Thain, University of Notre Dame
Rob Gardner, University of Chicago
John Hover, Brookhaven National Lab

http://virtualclusters.org

Lincoln Bryant, Jeremy Van, Benedikt Riedel, Robert Gardner, Jose Caballero, John Hover, Ben Tovar, and Douglas Thain,
VC3: A Virtual Cluster Service for Community Computation,
PEARC 2018. DOI: 10.1145/3219104.3219125
You have developed a large scale workload which runs successfully at a University cluster.

Now, you want to migrate and expand that application to national-scale infrastructure. (And allow others to easily access and run similar workloads.)

Traditional HPC Facility  Distributed HTC Facility  Commercial Cloud
Concept: Virtual Cluster

- 200 nodes of 24 cores and 64GB RAM/node
- 150GB local disk per node
- 100TB shared storage space
- 10Gb outgoing public internet access for data
- CMS software 8.1.3 and python 2.7
How do we get complex software delivered and deployed to diverse computing resources?

(without bothering sysadmins)
Delivery vs Deployment

• Delivery: Articulating and installing all of the necessary components at one site.

• Deployment: Moving all of the necessary components to each individual cluster node in an efficient manner.
Example: CMS Analysis Software

- Large Hadron Collider
- Compact Muon Solenoid
- Worldwide LHC Computing Grid
- Online Trigger

Many PB Per year
100 GB/s
Example: CMS Analysis Software

- Developed over the course of decades by 1000s of contributors with different expertise.
- Core codes in F77/F90/C99/C++18 + shell scripts, perl and python, scripts, shared libraries, config files, DSLs…
- Centrally curated by experts at CERN for consistency, reproducibility, etc.
- One release: 975GB, 31.4M files, 3570 dirs.
- Releases are very frequent!
Example: MAKER Genome Pipeline

- **Control Files**
- **Reference Libraries**
  - Protein Fastq
  - EST Fastq
  - Repeat DB
  - MB-GB in Size
- **Query**
  - 100-1000s of Contigs
- **Annotated Gene Model**
- **Evidence Files**
- **Execution Pipeline**
  - Align C1
  - Mask C1
  - Ann. C1
  - Align C2
  - Mask C2
  - Ann. C2
  - Align C3
  - Mask C3
  - Ann. C3
- **Annotation Steps of Single Contig**
Example: MAKER Genome Pipeline

• Large number of software dependencies (OpenMPI, Perl 5, Python 2.7, RepeatMasker, BLAST, several Perl modules)
• Composed of many sub-programs written in different languages (Perl, Python, C/C++)
• 21,918 files in 1,757 directories
• Typical installation model: Ask author for help!
Software Deployment/Delivery

• **Filesystem Methods**
  – Big Bucket of Software!
  – MetaFS: Metadata Acceleration
  – CVMFS: A Global Filesystem

• **Packaging Methods**
  – VC3-Builder: Automated Package Installation
  – Builder + Workflows

• **Container Methods**
  – Container Technologies
  – Containers + Workflows
Big Bucket of Software!

• Collect everything – binaries, interpreters, libraries – into one big tarball.

• Delivery is easy: copy, unpack, setenv.
  – (Not all software can be relocated to a new path)

• User-compatible approach – no sysadmin support needed, occupies user storage, etc.

• Just set up batch jobs to refer to the deployed location, set PATH, and go.
But: Metadata Storms!

• Common behavior: long burst of metadata access at the beginning of an application:
  – Search through PATH for executables.
  – Search through LD_LIBRARY_PATH for libraries.
  – Load Java classes from CLASSPATH.
  – Load extensions from file system.
  – Bash script? Repeat for every single line!

• Complex program startup can result in millions of metadata transactions!
Same problem on any parallel filesystem: Ceph, HDFS, Panasas, Lustre, …
MAKER Metadata Storm

Single Node Filesystem Load

Idea: Bulk Metadata Distribution

• We know some things in advance:
  – Which nodes need to load the software.
  – Which software is needed.
  – Software won't change during the run.

• Idea:
  – Build up all the metadata needed in advance.
  – Deliver it in bulk to each node.
  – Cache it for as long as the workflow runs.
Software metadata is cached on all nodes for the duration of the workflow and served at $$$ speed.
CVMFS Filesystem on >100K Cores Around the World

CMS Software 967GB File System at CERN

Generate Index

CVMFS: Cern-VM Filesystem

Proxy Cache Network

Individual Files

Metadata And Checksums

A

B

C

CVMFS App

FUSE

CVMFS

FUSE

CVMFS

App

 $$$

 $$$

 $$$

 $$$

 $$$

 $$$

 $$$
Nearly 2.5M metadata ops to start application

Reduced to a load of a single 147MB metadata file.

<table>
<thead>
<tr>
<th></th>
<th>Time</th>
<th>FUSE Syscalls</th>
<th>CernVM-FS Client Ops</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>stats (x1000)</td>
<td>opens (x1000)</td>
</tr>
<tr>
<td>CMS Software</td>
<td>12m05s</td>
<td>2429</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>8m14s</td>
<td>2429</td>
<td>11</td>
</tr>
<tr>
<td>Firefox</td>
<td>16s</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>2s</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>LaTeX</td>
<td>23s</td>
<td>150</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>17s</td>
<td>150</td>
<td>2</td>
</tr>
</tbody>
</table>

However CVMFS on HPC is tricky!

• Mounting filesystem on user nodes
  – FUSE -> requires some degree of privilege
  – Parrot -> requires precise ptrace behavior

• Live network access can be a problem.
  – Cache software in advance locally.
  – But which parts are needed for job X?

• CVMFS itself can be metadata intensive!
  – One site: Admins limited number of in-memory inodes allocatable by a given user, couldn't run!
Software Deployment/Delivery

• **Filesystem Methods**
  – Big Bucket of Software!
  – MetaFS: Metadata Acceleration
  – CVMFS: A Global Filesystem

• **Packaging Methods**
  – VC3-Builder: Automated Package Installation
  – Builder + Workflows

• **Container Methods**
  – Wharf: Docker on Shared Filesystems
  – Containers + Workflows
User-Level Package Managers

• Idea: Provide build recipes for many packages.
• Build software automatically in user space, each package in its own directory.
• Only activate software needed for a particular run.  (PATH, LD_LIBRARY_PATH,…)

• Examples:
  – Nix – Build from ground up for reproducibility.
  – Spack – Build for integration with HPC modules.
  – VC3-Builder – Build via distributed resources.
MAKER Bioinformatics Pipeline
VC3-Builder Architecture

Software Recipes
Upstream Sources
Builder
Cached Sources
Install Tree
Sealed Package
Recipe
A
B
C
D
Archival or Disconnected Operation
PATH
PYTHONPATH
LD_LIBRARY_PATH
Task
Task Sandbox
"vc3-builder –require ncbi-blast"

..Plan: ncbi-blast => [ ]
..Try: ncbi-blast => v2.2.28
....Plan: perl
....Try: perl
....could not add any source for:
....Try: perl
....could not add any source for:
....Try: perl
......Plan: perl-vc3-modules
......Try: perl-vc3-modules
......Success: perl-vc3-modules v0.1.0
....Success: perl v5.24.0
....Plan: python
....Try: python
....could not add any source for:
....Try: python
......Plan: openssl
processing for ncbi-blast-v2.2.28
preparing 'ncbi-blast' for x86_64/redhat6
Download 'ncbi-blast-2.2.28+-x64-linux.tar.gz' from http://download.virtualclusters.org/details: /tmp/test/vc3-root/x86_64/redhat6/ncbi-blast/v2.2.28/ncbi-blast-build-log

(New Shell with Desired Environment)

bash$ which blastx
/tmp/test/vc3-root/x86_64/redhat6/ncbi-blast/v2.2.28/bin/blastx

bash$ blastx -help
USAGE
blastx [-h] [-help] [-import_search_strategy filename]

bash$ exit
Problem: Long Build on Head Node

• Many computing sites limit the amount of work that can be done on the head node, so as to maintain quality of service for everyone.

• Solution: Move the build jobs out to the cluster nodes. (Which may not have network connections.)

• Idea: Reduce the problem to something we already know how to do: Workflow!

• But how do we bootstrap the workflow software? With the builder!
vc3-builder
  --require makeflow
  --require ncbi-blast
  --
  makeflow -T condor blast.mf
Bootstrapping a Workflow

Builder

Software Recipes
Upstream Sources

Makeflow
Build Makeflow
Build BLAST

BLAST Task
BLAST Task
BLAST Task
BLAST Task
BLAST Task
BLAST Task
Example Applications

Delivering a Global Filesystem with VC3-BUILDER
"vc3-builder –require cvmfs"

..Plan:  cvmfs => [, ]
..Try:   cvmfs => v2.0.0
....Plan:  parrot => [v6.0.16, ]
....Try:   parrot => v6.1.1
......Plan:  cctools => [v6.1.1, ]
......Try:   cctools => v6.1.1
........Plan:  zlib => [v1.002, ]
........Try:   zlib => v1.2.8
........Success: zlib v1.2.8 => [v1.2.0, ]
......Fail-prereq: cctools-v6.1.1
........Plan:  perl => [v5.010.000, v5.010001]
........Try:   perl => v5.10.0
..........Plan:  perl-vc3-modules => [v0.001.000, ]
..........Try:   perl-vc3-modules => v0.1.0
..........Success: perl-vc3-modules v0.1.0 => [v0.1.0, ]
..........could not add any source for: perl v5.010 => [v5.10.0, v5.10001.0]
..........Try:   perl => v5.16.0
..........Plan:  perl-vc3-modules => [v0.001.000, ]
..........Try:   perl-vc3-modules => v0.1.0
..........Success: perl-vc3-modules v0.1.0 => [v0.1.0, ]
..........could not add any source for: perl v5.016 => [v5.10.0, v5.10001.0]
..........Try:   perl => v5.24.0
..........Plan:  perl-vc3-modules => [v0.001.000, ]
..........Try:   perl-vc3-modules => v0.1.0
..........Success: perl-vc3-modules v0.1.0 => [v0.1.0, ]
..........Success: perl v5.24.0 => [v5.10.0, v5.10001.0]

(New Shell with Desired Environment)

bash$  Is /cvmfs/oasis.opensciencegrid.org

atlas  csiu  geant4  ilc  nanohub  osg-software  auger  enmr  glow  ligo  nova  sbgrid
cmssoft  fermilab  gluex  mis  osg
snoplussnolabca

...  

bash$  exit
Software Deployment/Delivery

• Filesystem Methods
  – Big Bucket of Software!
  – MetaFS: Metadata Acceleration
  – CVMFS: A Global Filesystem

• Packaging Methods
  – VC3-Builder: Automated Package Installation
  – Builder + Workflows

• Container Methods
  – Container Technologies
  – Containers + Workflows
Many Possible Container Techs

- ✔ Widely used
- ✔ Convenient global repo
- ❌ Builds up images locally
- ❌ Root Daemon

- ✔ Built on Docker Images
- ✔ No Root Daemon
- ❌ Requires Very Modern Kernel

- ✔ No Root Daemon
- ✔ Only one file
- ✔ Works with many image types
- ❌ Loop Devices
"runos slc6 – mysim.exe"
Desired Architecture
Types of Data

OS

Read-Only

Workdata
Kyle Sweeney and Douglas Thain,
Efficient Integration of Containers into Scientific Workflows,
Science Cloud Workshop at HPDC, June, 2018. DOI: 10.1145/3217880.3217887
Experiment: BLAST

We delivered 1/3rd less data, and finished in ~3/4ths the runtime using dynamic composition
Putting it All Together
Request 128 nodes of 16 cores, 4G RAM, 16G disk with RHEL6 operating system, CVMFS and Maker software installed:
Same Thing, Different Site:
Request 128 nodes of 16 cores, 4G RAM, 16G disk with RHEL6 operating system, CVMFS and Maker software installed:
• Big Bucket of Software
  + Maximum portability, compatibility, archivability.
  - Horrible metadata performance.
  + / - Correct with metadata oriented filesystems.

- User-Level Package Managers
  + Explicit statement of dependences. (repro!)
  + Deliver only needed components. (sharing!)
  - Long build/deploy processes. (use cluster)

• Container Technologies
  + Leverage commodity software tools.
  + Naturally metadata efficient.
  - Requires privileges, kernel tech, specialized tools.
  - Create new storage management problems.
Thoughts on Dependencies:

• Make software dependencies more explicit.
  – Proposed: Nothing should be available by default, all software should require an "import" step.

• Need better, portable, ways of expressing:
  – What software environment the user wants.
  – What software components are actually used.
  – What environment the site provides.

• The ability to nest environments is critical!
  – Sysadmin provisions machine via VM/container.
  – Batch system provisions slot with container.
  – User provisions software with container.
Thoughts on Filesystems

• Open/read/write/close has worked well for a long time, but seems to be too small a granularity for large scale systems/software.

• Can we have flexible transaction to balance between small changes and wide distribution?

• Do we need new filesystem ops?
  – `fd = Opentree("/home/dthain",O_RDONLY);`
  – `Results = Search("$PATH","sim.exe");`
  – Something like SQL for metadata?
Acknowledgements

Notre Dame CMS:
Kevin Lannon
Mike Hildreth
Kenyi Hurtado

Univ. Chicago:
Rob Gardner
Lincoln Bryant
Suchandra Thapa
Benedikt Riedel

Brookhaven Lab:
John Hover
Jose Caballero

People in the Cooperative Computing Lab

Douglas Thain
Director

Benjamin Tovar
Research Soft. Engineer

Peter Ivie

Nicholas Hazekamp

Charles Zheng

Nathaniel Kremer-Herman

Tim Shaffer

Kyle Sweeney

DE-SC0015711
VC3: Virtual Clusters for Community Computation

ACI-1642409
SI2-SSE: Scaling up Science on Cyberinfrastructure with the Cooperative Computing Tools