VC3: A Virtual Cluster Service for Community Computation

Lincoln Bryant
University of Chicago
Chicago, Illinois
lbryant@uchicago.edu

Jeremy Van
University of Chicago
Chicago, Illinois
jeremyvan@uchicago.edu

Benedikt Riedel
University of Chicago
Chicago, Illinois
briedel@uchicago.edu

Robert W. Gardner
University of Chicago
Chicago, Illinois
rwg@uchicago.edu

Jose Caballero Bejar
Brookhaven National Lab
Upton, New York
jcaballero@bnl.gov

John Hover
Brookhaven National Lab
Upton, New York
jhover@bnl.gov

Ben Tovar
University of Notre Dame
Notre Dame, Indiana
btovar@nd.edu

Kenyi Hurtado
University of Notre Dame
Notre Dame, Indiana
khurtado@nd.edu

Douglas Thain
University of Notre Dame
Notre Dame, Indiana
dthain@nd.edu

ABSTRACT

A traditional HPC computing facility provides a large amount of computing power but has a fixed environment designed to satisfy local needs. This makes it very challenging for users to deploy complex applications that span multiple sites and require specific application software, scheduling middleware, or sharing policies. The DOE-funded VC3 project aims to address these challenges by making it possible for researchers to easily aggregate and share resources, install custom software environments, and deploy clustering frameworks across multiple HPC facilities through the concept of "virtual clusters". This paper presents the design, implementation, and initial experience with our prototype self-service VC3 platform which automates deployment of cluster frameworks across diverse computing facilities. To create a virtual cluster, the VC3 platform materializes a custom head node in a secure private cloud, specifies a choice of scheduling middleware, then allocates resources from the remote facilities where the desired software and clustering framework is installed in user space. As resources become available from scheduled nodes from individual clusters, the research team simply sees a private cluster they can access directly or share with collaborators, such as a science gateway community. We discuss how this service can be used by research collaborations requiring shared resources, specific middleware frameworks, and complex applications and workflows in the areas of astrophysics, bioinformatics and high energy physics.

CCS CONCEPTS

• **Computer systems organization → Grid computing**;

KEYWORDS

OSG, distributed data access, CVMFS, XENON, EGI, XENON1T, XSEDE, Campus Cluster, HTCondor, Distributed computing, Cloud computing, Grid computing

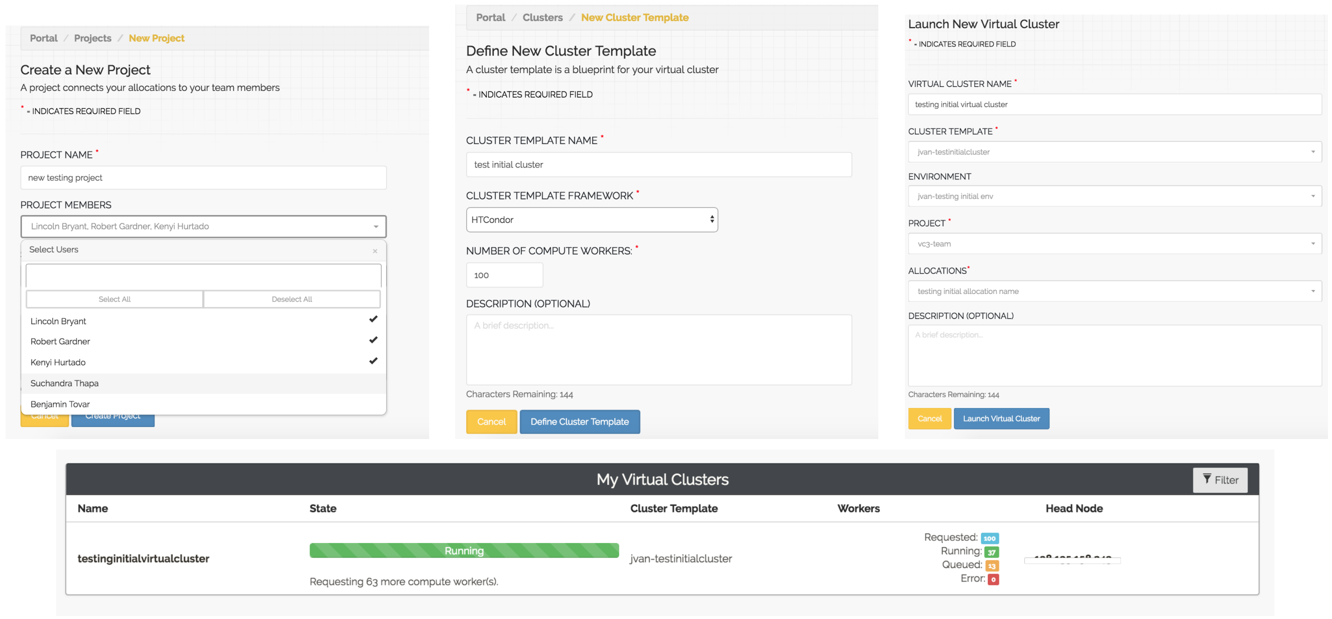
ACM Reference format:
Lincoln Bryant, Jeremy Van, Benedikt Riedel, Robert W. Gardner, Jose Caballero Bejar, John Hover, Ben Tovar, Kenyi Hurtado, and Douglas Thain. 2018. VC3: A Virtual Cluster Service for Community Computation. In *Proceedings of Practice and Experience in Advanced Research Computing, Pittsburgh, PA, USA, July 22–26, 2018 (PEARC '18),* 8 pages.
https://doi.org/10.1145/3219104.3219125

1 INTRODUCTION

The traditional high performance computing (HPC) model assumes that users conduct their work at one of a small number of facilities, each equipped with advanced computing and storage capabilities. Professional system administrators procure, install, operate, and upgrade the facility according to local needs, working with users to adjust applications so that they meet local conditions. As a result, traditional HPC sites differ from each other in many ways small and large, including details like the operating system, the file storage system, the batch queue, the applications installed, and more.

Because of these local differences, complex applications constructed in a given facility are often difficult to disentangle from the peculiarities of that facility. Production applications are often complex assemblies of multiple scripts, libraries, and executables dependent on system services such as middleware, databases, caches, and other internet-accessible resources. Constructing such an assembly in one location is hard enough; replicating it at another facility can be a dreaded task for even a sophisticated user or system administrator. As a result, powerful computing facilities may be underutilized, and opportunities for research missed simply because deploying a complex system is too time consuming.

This is not a problem for a single user at one facility, but it presents a serious barrier to multi-institution collaboration, extension, and replication of research. Researchers have access to different selections of resources according to their local conditions. For example, university researchers might have a fair-share slice of


Portal / Projects / New Project

Create a New Project

A project connects your allocations to your team members

* - INDICATES REQUIRED FIELD

PROJECT NAME *

new testing project

PROJECT MEMBERS

Lincoln Bryant, Robert Gardner, Kenyi Hurtado

Select Users

Select All | Deselect All

Lincoln Bryant ✓
Robert Gardner ✓
Kenyi Hurtado ✓
Suchandra Thapa
Benjamin Tovar

Cancel | Create Project

Portal / Clusters / New Cluster Template

Define New Cluster Template

A cluster template is a blueprint for your virtual cluster

* - INDICATES REQUIRED FIELD

CLUSTER TEMPLATE NAME *

test initial cluster

CLUSTER TEMPLATE FRAMEWORK *

HTCondor

NUMBER OF COMPUTE WORKERS: *

100

DESCRIPTION (OPTIONAL)

A brief description...

Characters Remaining: 144

Cancel | Define Cluster Template

Launch New Virtual Cluster

* - INDICATES REQUIRED FIELD

VIRTUAL CLUSTER NAME *

testing initial virtual cluster

CLUSTER TEMPLATE *

jvan-testinitialcluster

ENVIRONMENT

jvan-testing initial env

PROJECT *

vc3-team

ALLOCATIONS*

testing initial allocation name

DESCRIPTION (OPTIONAL)

A brief description...

Characters Remaining: 144

Cancel | Launch Virtual Cluster

My Virtual Clusters

Filter

Name	State	Cluster Template	Workers	Head Node
testinginitialvirtualcluster	Running Requesting 63 more compute worker(s).	jvan-testinitialcluster	Requested: 100 Running: 37 Queued: 13 Error: 0	[redacted]

Figure 1: User Interface to VC3

The user interacts with the VC3 web portal (`virtualclusters.org`*) to register an allocation for a computing resource, define a project and collaborators, configure a software environment, launch a virtual cluster, and monitor its current state. Once running, all members of the project can log into the head node of the VC and make use of the cluster via the installed middleware.*

a university cluster, a peer-reviewed allocation on a national computing facility like XSEDE [33] or Open Science Grid (OSG) [27], or a pay-as-you-go arrangement with a commercial cloud provider like Amazon or Google. Multiple researchers working together may not necessarily have access to a common subset of facilities, but they may wish to pool their combined resources together.

We address this problem through the concept of **virtual clusters** (VCs) overlaid on existing computing resources. The key idea is to make use of existing facilities in their current state, and inject into them (as an unprivileged user job) a thin layer of software which surveys the available resources, delivers missing dependencies, and then deploys the research team's desired middleware or application structure on top. The research team supplies the VC3 platform with a set of facilities where they hold allocations, a middleware selection, and a software environment. The VC3 service then provisions the resources across the desired computing sites using only individuals' login accounts and local allocations, providing the team with virtual cluster headnode and login server in a secure private cloud.

In this paper, we describe the concept and architecture of our virtual cluster service, along with a first prototype of the concept (`virtualclusters.org`). We demonstrate that the prototype is able to bootstrap and execute complex applications (South Pole Telescope, XENON1T, Lobster, SHRiMP) using standard middleware (HTCondor, CVMFS, Pegasus, Makeflow, Work Queue) running at scale on infrastructure that was not otherwise prepared for the application.

2 THE VIRTUAL CLUSTER CONCEPT

The concepts behind the VC3 platform come from the experiences of distributed high-throughput computing on infrastructure such as the OSG. In 2013, we developed the CI Connect [3] service which provided a "virtual cluster" experience for researches submitting applications and workflows to the OSG and dedicated campus HPC clusters. The service, which leverages the capabilities of HTCondor [26] (and derivative "glidein" and "overlay" concepts [29]) integrated resource targets in which scientific collaborations held allocations while providing access to the shared (and preemptable) OSG ecosystem. These services, which included OSG Connect [8], have shown to be highly effective tools for over 200 projects, supporting over one thousand researchers and delivering over 200 million CPU core-hours for science.

The VC3 platform automates creation of such virtual cluster environments but with generalizations for the clustering layer and sharing model. Its primary functions are:

- To bind allocations from individual investigators across institutional HPC clusters for shared projects supporting multi-institutional scientific collaboration.
- To provision middleware (or clustering framework) over heterogeneous HPC resources while preserving local setups, resource scheduler systems, policy and autonomy.
- To build a uniform software environment across multiple resource targets using preinstalled binaries (if available) or downloading from a repository, according to the researcher's package specification.
- To size the cluster dynamically according to user specifications.

Note this similar to service offerings from commercial cloud providers, such as Google which offers tools for constructing dynamically sized HTCondor clusters [4], and more recently, SLURM clusters [5]. However, in this case, the VC3 platform is building equivalent compute environments from allocations on institutional computing centers and national scale HPC facilities.

Figure 1 shows how a researcher interacts with the VC3 service. We leverage the Globus Auth[34] service from the Globus Platform [12] to provide a portal infrastructure and access to institutional identity management services. The end user interactions with the VC3 service consist of establishing an account and profile, registering an allocation from an institutional HPC cluster resource, creating a project to which the allocation may be associated, inviting research team members to join the project, creating a virtual cluster template (middleware type, software environment, number of worker nodes), and then launching a virtual cluster instance, the head nodes of which are provisioned by the VC3 platform in a secure private cloud. Once provisioned, research team members may ssh login to the head node service to launch their workflows.

3 SYSTEM ARCHITECTURE

Figure 2 shows the architecture of the VC3 service. There are three tiers of services involved:

- The **static infrastructure** contains the core VC3 processes that interact with the end user and implement the fixed points of the overall service.
- The **dynamic infrastructure** consists of the virtual machines and processes that are provisioned to create each virtual cluster.
- The **remote resources** are the existing computing facilities (campus clusters, HPC centers, commercial clouds, etc) that VC3 is designed to harness.

To provision a virtual cluster, the VC3 service must create a suitable head node in the dynamic infrastructure, then create a number of worker nodes in the remote resources.

The front end of VC3 is very similar to that of a conventional web service: a stateless **web server** interacts with the client and manipulates state within a **database**. Because the actions performed by VC3 are potentially expensive and long running, they are carried out in the background by a **master** process, which reads the intent of the user to create or modify a virtual cluster, and then triggers actions to move towards the desired state. For each virtual cluster to be provisioned, the master process creates a **head node** in one resource, and commissions a queue in the **factory** which provisions multiple **worker nodes** in one or more remote resources.

The steps to setup and teardown each virtual cluster are described by a finite state machine, where each edge represents a discrete interaction with another local or remote service, such as starting a process, invoking a head node, and so forth. The master periodically queries the database for virtual cluster requests whose desired state does not match the actual state, and then triggers the appropriate action in the finite state machine. Because a large number of virtual clusters may exist at once, the master has a number of rate-limiting controls to prevent excess consumption of local resources.

A **factory** is responsible for managing the dynamic set of worker nodes needed by a given virtual cluster. The factory continuously matches the number of actual running worker nodes to the policy given by the end user during the lifetime of the VC. When more workers are needed, they are provisioned by submitting worker jobs to remote computing resources, and then removed if no longer needed. While easily stated, the task is quite complex because of the independent behavior of the remote sites. While the desired case is for workers to execute immediately, the jobs could be rejected, delayed, migrated, or evicted, all due to remote policies and dynamic load. The factory must take great care to advance the system state gradually while avoiding repeated actions that would have the effect of a denial of service attack on the computing resource.

Of course, the end user does not want a simple bare cluster, but a functioning system with working software and operating services to accomplish a particular workload. To this end, both head and worker nodes execute a **builder** process that installs the necessary software environment (more on that below) and then invokes the desired middleware components. Most middleware fits the pattern of a **middleware worker** that runs on the worker nodes of the VC, and a **middleware master** that runs on the head node of the VC.

For example, when HTCondor [31] is selected as a middleware, an HTCondor **startd** is run on the VC workers, while the VC head node runs an HTCondor **collector**, **negotiator**, and **schedd**. If Spark [35] is selected as a middleware, then a Spark **worker** runs on each VC worker and the Spark libraries are installed on the head node. When the user starts a Spark **driver** on the head node, the workers contact it and begin to serve requests. In a similar manner, when the Work Queue [11] middleware is selected, the Makeflow [10] software is installed on the head node, and waits for Work Queue **workers** on the VC workers to contact it.

4 RESOURCE MANAGEMENT WITH APF

To handle submission of middleware workers, VC3 uses AutoPyFactory (APF)[19]. Developed at Brookhaven National Laboratory, APF began as the component used to manage pilot job submission for the ATLAS physics experiment. APF was adapted to be used in VC3 via its modular, plugin-based design, together with flexible configurations descriptions. Its use of HTCondor-C as a generic external interface allows APF to target any platform that HTCondor supports (Grid Compute Elements, SSH-accessible resources, and cloud targets like AWS or Openstack).

APF runs as a single daemonized process, launching separate threads to handle each worker nodes queue, shared internal functions, and global factory functions. An example of global functions are threads to export log directories via HTTP, a utility to maintain authentication tokens, and periodic factory reconfiguration. An example of a service used by multiple queues is the plugin to query the local HTCondor-C schedd persistent state.

Each of the queues normally manage interactions with a single resource target. The behavior of the queues is determined by the combination of a set of plug-ins, invoked in a fixed order, in a loop, each one in charge of the performance of a well defined action. These plugins allow composition to define simple algorithms for determining the queues behavior, e.g. a MaxPerCycle plugin limits

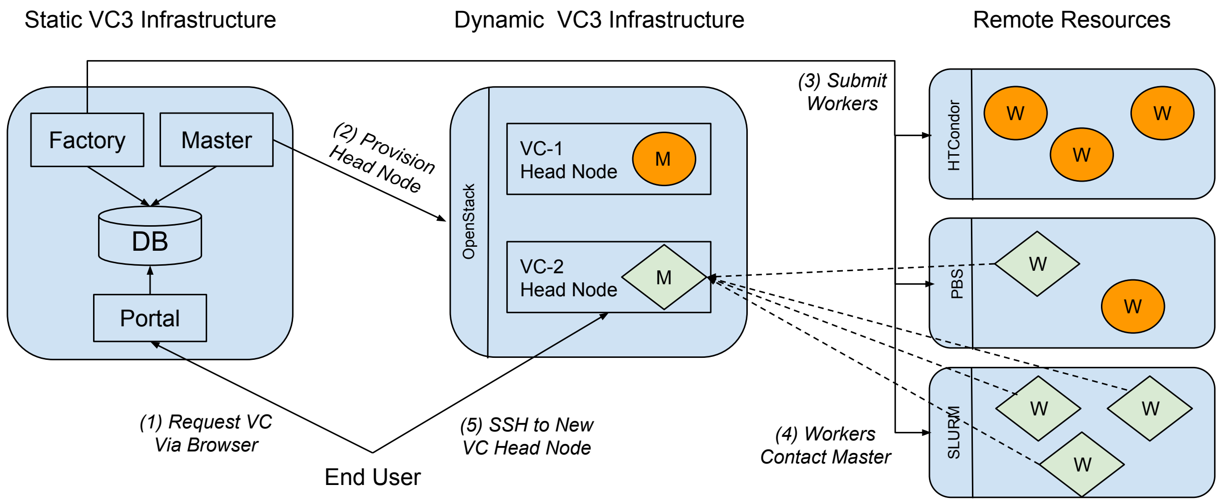


Figure 2: VC3 System Architecture

An example of the VC3 infrastructure running two virtual clusters, each running a different middleware, represented by circles and diamonds, respectively. To procure a virtual cluster: (1) The end user requests a VC via the web portal, which records the request in the database. (2) The master instantiates a head node and starts a middleware-master there. (3) The factory selects remote resources and submits worker jobs there. (4) The worker jobs start the middleware-workers, which contact the middleware-master on the head node. (5) The user connects to the head node and uses the cluster.

how many new overlay jobs are submitted per cycle; a MaxPending plugin defines when the queue will cease submission.

In the VC3 context, APF was modified with an additional configuration plugin to periodically retrieve and update its overall configuration from the VC3 database, which is in turn periodically updated by the VC3 master process based on current virtual cluster requests. A VC3-specific monitor plugin was also implemented to communicate both queue information and overall factory information back to the database, for display via the web portal.

Work was done to add full remote installation management to the ability of HTCondor-C to submit via SSH to remote resources. An SSH-specific submit plugin was written to ensure that before attempting to submit to a resource, APF ensures that the HTCondor-C glide-in software needed to manage jobs is installed in the user's home directory. If not, it performs the installation and only then proceeds with overlay job submission.

In a typical virtual cluster invocation, the user will have authorization to use multiple target resources to satisfy the request. In the current implementation, the VC3 master process performs a simple static distribution of overlay jobs across the resources to be utilized. In the near future, APF will allow the VC3 master process to provide a configuration that defines more complex strategies. E.g., load balancing across a set of resources, and ordered fill (where a second resource is only used when an initial resource is full). The APF configuration mechanism will allow arbitrarily nested sets of load balanced and/or ordered fill resource targets.

Both load balancing and ordered fill fundamentally depend on understanding what it means for a resource to be full. This is not a trivial definition, because jobs submitted to a site may wait in the queue for a long time. However, due to local priorities, just because a long queue exists does not mean the jobs will not run right away. For our purposes, 'full' can be defined in terms of how many new overlay jobs have started within a given interval in the past, e.g. resource A is full if no more than 2 new jobs have started in the last 30 minutes. This functionality can be fully parametrized such that each resource can have distinct, customized definitions of full. These definitions will be part of the curated properties for each VC3 resource, so that users need not provide them.

5 SOFTWARE ENVIRONMENTS

The end user of VC3 expects a precise software environment in order to successfully run both applications as well as the desired middleware. The most comprehensive way to accomplish this would be to require the user to construct a suitable disk image with all of the required software, and then deploy it on all nodes, using either virtual machine or container technologies. (This is precisely what happens in a cloud infrastructure.)

However, such an approach is neither feasible nor desirable in VC3. The remote computing resources generally restrict end users to operating as normal unprivileged users in a shared filesystem, where a specific operating system and selection of applications has been chosen that services the needs of the local community and works well with the local computing resources. Although some sites allow for the deployment of container images, this is by no means universally supported, and there is no agreement on which of the available container technologies to use: Docker [1], Singularity [25], CharlieCloud [28], and Shifter [24] are all in use to varying degrees. Further, deploying a container may even by desirable or counterproductive when the native environment is already suitable for the user's task.

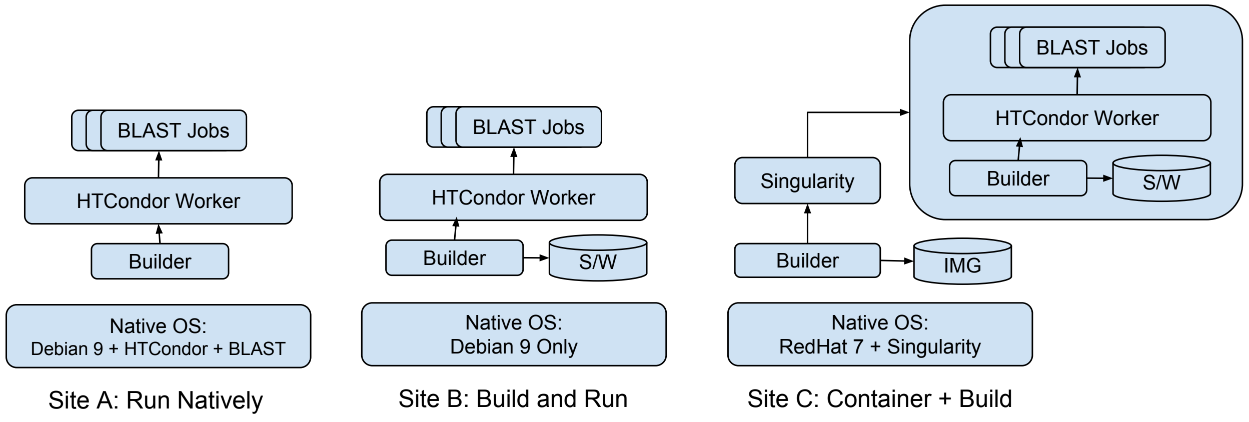


Figure 3: Deploying an Environment on three Sites

Multiple techniques may be needed to provision a common environment – operating system, middleware, and application software – across multiple sites. The VC3 builder examines the local environment and provides whatever is needed to repair to the desired state. In this example, a user wishes to deploy Debian 9, HTCondor, and BLAST across multiple sites. At Site A, the environment is already present, so the builder simply invokes the middleware directly. At Site B, the operating system is compatible, but the builder must download and install the middleware and application. At Site C, the OS is not compatible, so the builder deploys a container in which the middleware and application must also be built.

To work within these requirements, VC3 asks the user to specify a computing **environment** with any of these elements:

- A base operating system type and version (e.g. Redhat-6, CrayOS-9, Debian-11)
- A container name, drawn from Docker Hub.
- A list of required software systems (e.g. python-2.7, blast-2.2.28, lapack-3.7.0)

At each computing resource, VC3 provisions the necessary elements by whatever means are available. If the native environment is compatible with the desired environment, then it is used directly. If it is not compatible (or not complete) then the VC3 **builder** [32] will construct the necessary environment, using whatever tools are available at the local site. The builder is simply invoked as a wrapper around the worker to be deployed, and is given the name of the target operating system and software dependencies required.

Although the builder is hidden from the end user in VC3, it can be invoked as a standalone tool. For example, to execute a shell in a Debian-9 OS environment with the Spark middleware and BLAST application installed, the builder is invoked like this:

```
vc3-builder --require-os debian9
            --require spark
            --require blast
            --interactive
            -- bash
```

In this example, the builder will evaluate whether the local operating system is Debian 9. If so, it is used natively. If not, then it looks for an installed container technology, and downloads a container image corresponding to Debian 9, and enters it. In either case (native or container), the user's PATH is examined for the Spark and Blast software. If present, it is used natively, if not, it is downloaded and installed into the user's home directory.

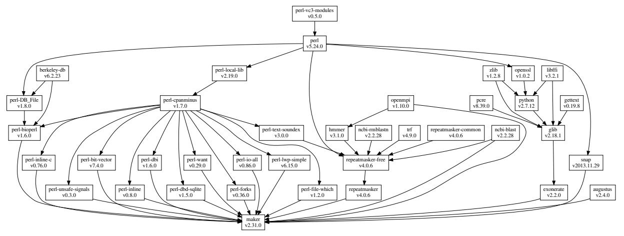


Figure 4: MAKER Software Dependencies

To support these software builds, a global master repository contains a set of package *recipes* expressed in JSON. Each recipe indicates the source packages, build commands, import procedure, and dependencies of a given software page. For each software package requested, a recipe and its dependencies are located and constructed into a dependency graph. For production software, these graphs can be quite complex, as can be seen in Figure 4 for **MAKER** [14], a genome annotation pipeline used in the bioinformatics community.

For each dependency in the graph, builder checks to see if the package is already available natively. If not, then the necessary source packages are downloaded from *upstream sources* into a *cache directory*. Each package is built and installed into the *install tree* which is by default inside the user's home directory.

To facilitate bootstrapping, the builder is designed to have as few dependencies as possible that can be compiled to a truly static binary which includes the packages recipes. This allows builder to be deployed as a single executable file.

When a shared filesystem is available, the builder can run parallel builds across different computational nodes. In this mode, an instance of the builder computes the steps necessary to install all the required packages, and according to the batch system available, it queues and checks the progress of additional builder instances that perform the actual installation. Not only does this accelerate build times, but it also moves computation off the head-nodes where runtimes may be restricted by policy.

6 SECURITY CONSIDERATIONS

Great care is required in the design and implementation of VC3 in order to protect access to each user's credentials and the resources that they may access. To the extent possible, VC3 relies on standard tools to implement authentication and authorization.

In the static infrastructure, all components communicate via HTTPS with the VC3 master serving as a certificate authority. In order for a VC3 component to communicate with other components, it must have a certificate issued by the VC3 master. In addition, all internal components are firewalled to the outside world.

Passwords are never handled directly by VC3 in any form. End users make use of Globus Auth to authenticate to the web portal, while cluster and dynamic infrastructure authentication makes use of standard SSH public keys. By design, VC3 never handles private SSH keys owned by end-users. The only use of private SSH keys is for internally-generated keys created for each head node in the dynamic infrastructure, and these never leave the VC3 system. To access a head node, users upload their SSH public key to the service via the web portal. This key is automatically placed into the user's home directory on the dynamic head node at cluster installation time, at which point users can log in and begin to use their VC.

To manage authentication at the other end of the system, where VC3 communicates with a remote resource, a user receives an SSH public key generated by the web portal and is required to add that that key to their SSH authorized keys on the resource. In some systems, this requires an additional step of adding the key to a web portal or other system owned by the resource. Once the key has been added, the VC3 system will validate that it is able to log in to the remote resource.

This approach is made more complex by the emerging requirement of multi-factor authentication, which requires that users present a second login confirmation by cell phone or other device. At some facilities, VC3 has been granted an "MFA Exemption" that treats a limited source IP range (such as the VC3 factory subnet) as one factor of the multifactor authentication process. In more restrictive facilities, we have developed a standalone version of our factory service that runs on the head node of the site and pulls work from the VC3 system. In this way we can still provide virtual cluster services to systems where automation is restricted.

7 EXAMPLE APPLICATIONS

South Pole Telescope via HTCondor+CVMFS The South Pole Telescope (SPT) [21] is a microwave and millimeter-wave telescope located at the Amundsen–Scott South Pole Station in Antarctica that focuses on the observation of the Cosmic Microwave Background. The third generation of detectors [15] of SPT produce an order of magnitude more data than previous generations. With the increase in data rate, a proportionate increase computing resource requirements occurred. This increase in computing requirements initiate a shift for the collaboration from solely using collaboration and campus resources to use distributed computing resources, including OSG, National Energy Research Scientific Computing Center's (NERSC) Cori [6, 7], and campus clusters, for data processing and Monte Carlo production.

The collaboration is seeking a way to access their various resources through a common interface. In this case, we are using VC3 to extend SPT's infrastructure by adding the University of Chicago Research Computing Center (RCC) [9] and Cori to their already existing resource pool through OSG. A virtual cluster is instantiated with HTCondor middleware, with the head node configured to allow "flocking" from existing SPT OSG submission nodes. This allows users to transparently use RCC and Cori resources without significant changes to their existing job workflow.

Integrating the Cori resource required additional effort. The initial challenge was that Cori uses a custom version of the Cray operating system (OS). Currently, SPT distributes its software dependencies using CVMFS [18] for RHEL6, RHEL7, and their derivatives. We do not have the ability to build the entire software stack for the specific OS without requiring that every user has access to the machine and keep up with code changes. Instead of having every user compile the code for Cray OS, we used Shifter [24] and an OSG-provided Docker image to create a RHEL6 environment for the jobs to run in. This allows us to use the CVMFS software repository by syncing it to Cori's shared filesystem and bind-mounting the directory to location to the expected location for CVMFS, i.e. `/cvmfs`. With these changes, we were able to execute SPT workloads for both data processing and Monte Carlo production on Cori through their already-established OSG interface.

XENON1T Monte Carlo via Pegasus+HTCondor. The XENON experimental program [13] aims to study yet to be discovered exotic species of elementary particles called dark matter through their interactions with xenon atoms. The experiment is the third and current generation of detector consisting two tonnes of ultrapure liquid xenon inside a dual-phase (liquid-gas) xenon time projection chamber surrounded by a water-based muon veto.

The XENON1T experiment has significant computing resources available at campus clusters and grid sites The resources at grid sites were integrated through the OSG infrastructure. The campus clusters were integrated in the same manner as described above, i.e. the VC3 head node is configured to allow "flocking" from the existing XENON OSG submission node. The workflow itself was controlled through the Pegasus workflow management system [22], which transparently uses both the OSG and campus cluster resources without any additional configuration.

SHRiMP via Makeflow+Work Queue. SHRiMP is a tool used in bioinformatics for genome alignments. A common SHRiMP workflow involves thousands of alignment jobs against a target genome. We tested a SHRiMP workflow consisting of over 5000 jobs expressed as a Makeflow [10] workflow. Makeflow allows the execution of workflows expressed as direct acyclic graphs on clusters, clouds and grids using different batch systems. For our test, Makeflow ran on top of Work Queue [11] middleware.

We directed the VC3 portal to create a virtual cluster of 350 work queue workers. Once the headnode was available, the data for the SHRiMP workflow and its Makeflow specification was copied to it using **scp**. On the headnode, a Makeflow master process was launched that registered its address location in a globally available catalog, tagged with the name given to the virtual cluster at creation time. Using this tag, the workers connect to the globally available catalog to find the location of the master and connect to it. Once a worker connects, the master starts to deliver tasks to it, according to tasks that are available for execution. For this particular test, all the tasks in the workflow terminated successfully in close to 3 hours, which highlights the ease with which the VC3 system can provide with short-lived clusters for particular needs.

CMS Analysis via Lobster at NERSC and Notre Dame. The Compact Muon Solenoid (CMS) experiment [23] analyzes large amounts of data collected from high energy particle collisions produced at the Large Hadron Collider (LHC) at CERN. The level of CMS workload produced has a very high demand in terms of computing resources, which are traditionally handled in a uniform way on computing nodes integrated in the Worldwide LHC Computing Grid (WLCG) [16].

Lobster is a workflow manager that allows the harnessing of resources that are not part of the WLCG, assembling tasks on-the-fly for dynamic job sizing, providing the software environment needed and handling the stage out of job results remotely. It uses a master-worker architecture implemented with Work Queue and has been extremely successful for utilizing the opportunistic resources available in the Notre Dame campus cluster, submitting Work Queue workers via HTCondor and using Singularity to provide the right Operating System. Resources like NERSC, which work with a different container solution (Shifter) and with a different batch system (SLURM), make it challenging to migrate.

The VC3 service allows the construction of a Work Queue virtual cluster with the right Operating System on the Worker Nodes (via Shifter or Singularity when needed) that allows Lobster to use both Notre Dame campus resources and NERSC in a simple and transparent way that would otherwise not be possible.

8 CURRENT STATUS

Currently, the VC3 platform is available as a limited beta for science user groups that are working closely with the VC3 team. At present, it is possible to create, use, and destroy a virtual cluster from the web portal or command-line utility. Users visiting the VC3 portal are able to: add allocations to resources, define projects, create "cluster templates" describing the middleware type and scale for their virtual clusters, pick software environments available via the VC3 builder, and instantiate virtual clusters.

Today, users may create HTCondor[26] and Work Queue[17]-based virtual clusters. Users can additionally select software environment for their cluster head node and workers through the website, and the installation is handled via the VC3 builder. These environments include common programming runtimes and compilers such as Python, Scala, GCC, Perl, Tcl, etc. Where possible, users can request environment such as OSG OASIS[2], provided by the builder via Parrot[30]. We have also successfully tested Spark at a small scale by using the S3 data plugin (in lieu of a shared filesystem) and HTCondor middleware to distribute Spark workers. Currently Spark can be deployed as an environment, but we plan to support it as a first-class citizen middleware.

While we are not supporting custom VC3 installations at present, all of the software is fully available on GitHub, packed in RPM format, and tested thoroughly on the CentOS 7 platform.

9 CHALLENGES AND FUTURE WORK

Our experience so far has highlighted a number of challenges in the cyberinfrastructure domain that go beyond VC3 and would be worthy of further study.

Troubleshooting nested services. An increasing amount of cyberinfrastructure is constructed by deeply nesting multiple technologies from multiple providers. In VC3, there is a long chain from submission to execution: in one case, APF uses HTCondor-C to submit jobs to SLURM, where each job uses Singularity to run the builder to deploy HTCondor to run a job. When each of these components works as expected, jobs run normally. However, problem in any one component – whether due to user error, site configuration, network disruption, or something else – results in failures that are extremely hard to diagnose. Each system uses a different convention for indicating success or failure, and a different means of logging information. Uncovering the nature of the failure is a challenge for the systems expert, much less the end user. New techniques are needed to enable the construction and troubleshooting of these nested systems. Possible solutions might be conventions for invoking nested services, tools for verifying assertions deep within the stack, or the equivalent of a "backtrace" for detecting in which layer a failure occurred.

Fine grained accounting and resource control. Computing sites typically offer accounting and resource control at the level of a single user or funded project. For every gigbyte stored or cpu-hour consumed, the resource consumption is tracked, and then halted if the user or project completely consumes their allocation. However, VC3 now enables users to share resource allocations for projects of a smaller scope, perhaps just a certain analysis task, or a short-lived simulation campaign. In such an environment, users will not want to donate their *entire* allocation, but rather delegate some limited slice to the project. An external service like VC3 can provide some coarse controls on the use of this slice, but is not close enough to the resource to immediately halt any overage. A more effective solution would be for the facilities to allow users to dynamically create sub-allocations which were individually accounted for and enforced as children of the parent allocation. Then, users could provide these sub-allocations to outside parties with confidence that resources would not be overrun.

Curating technical knowledge of computing resources. While there is a general commonality of design among HPC facilities, the technical details needed to access and use across a distributed system are not easily discovered. A typical site consists of a small set of login nodes that provide access to workers through a batch system, and provide some kind of shared storage via a parallel or distributed filesystem. To use the batch system effectively, one must not only know the software in use (HTCondor, SLURM, PBS, etc) but also what details are unique to the local site, such as the names of queues or resource classes. Likewise, to use a large filesystem effectively, it is necessary to know what mount points represent home directories vs temporary data vs long-term project storage. Even details of the consistency semantics may matter: in order to implement a reliable workflow, it is necessary to understand whether the system provides strict metadata semantics, read-your-write semantics, or something else entirely. Some of the details may be documented, but often the documentation it out of date as the system evolves, leaving the job to a savvy professional to figure things out by hand. If it were possible to observe this information more automatically and then curate it for use by multiple projects would better enable multi-facility computing of all kinds.

Authentication models for automated computing. Much of our shared security infrastructure is built around the assumption that there is a human in the loop at a fine granularity, ready and willing to enter a password or approve a login. This assumption is baked into emerging practices such as multi-factor authentication, whereby the end user must not only provide a password but also approve a login via a cell phone or other device. While this is a perfectly sensible approach for interactive logins, it does not easily accommodate any kind of external automation, which is essential for the conduct of large scale science. New ideas are needed to understand how the security requirements of computing sites can be respected without requiring the user to interact with their cell phone on every job or file transfer. One recent approach (Reverse GAHP [20]) is to reverse the sense of the connection, so that the user logs into the site manually, and then directs a proxy to connect to an external workload manager. Another approach could be to reduce the scope of credentials so that they address only a narrow activity. For example, a user-signed certificate might grant authority only to transfer files between specific locations, or to run only a single kind of executable.

ACKNOWLEDGEMENTS

This work was supported by the Department of Energy NGNS program via grant DE-SC0015711. We are grateful to Suchandra Thapa (U.Chicago), Paul Brenner (U. Notre Dame), and John Gentile (Science Gateways Institute) for their assistance and advice.

REFERENCES

[1] [n. d.]. Docker Website. http://www.docker.com. ([n. d.]). Accessed: 2018-03-26.
[2] [n. d.]. OSG Application and Software Installation Service (OASIS). https://opensciencegrid.github.io/docs/worker-node/install-wn-oasis. ([n. d.]). Accessed: 2018-03-26.
[3] 2018. CI Connect Website. (2018). Retrieved March 21, 2018 from http://www.ci-connect.net
[4] 2018. Google HTCondor Website. (2018). Retrieved March 21, 2018 from https://cloud.google.com/solutions/high-throughput-computing-htcondor
[5] 2018. Google Large Scale Technical Computing Website. (2018). Retrieved March 21, 2018 from https://cloud.google.com/solutions/using-clusters-for-large-scale-technical-computing
[6] 2018. NERSC Cori Website. (2018). Retrieved 2018/03/21 from http://www.nersc.gov/users/computational-systems/cori
[7] 2018. NERSC Website. (2018). Retrieved 2018/03/21 from https://www.nersc.gov
[8] 2018. OSG Connect Website. (2018). Retrieved March 21, 2018 from http://osgconnect.net
[9] 2018. UChicago RCC Main Website. (2018). Retrieved March 15, 2018 from https://rcc.uchicago.edu/
[10] Michael Albrecht, Patrick Donnelly, Peter Bui, and Douglas Thain. 2012. Makeflow: A Portable Abstraction for Data Intensive Computing on Clusters, Clouds, and Grids. In *Workshop on Scalable Workflow Enactment Engines and Technologies (SWEET) at ACM SIGMOD.*
[11] Michael Albrecht, Dinesh Rajan, and Douglas Thain. 2013. Making Work Queue Cluster-Friendly for Data Intensive Scientific Applications. In *IEEE International Conference on Cluster Computing.*
[12] Rachana Ananthakrishnan, Kyle Chard, Ian Foster, and Steven Tuecke. 2015. Globus Platform-as-a-Service for Collaborative Science Applications. *Concurrency - Practice and Experience* 27, 1 (2015), 290–305.
[13] E. Aprile et al. 2011. Design and performance of the XENON10 dark matter experiment. *Astroparticle Physics* 34 (April 2011), 679–698. https://doi.org/10.1016/j.astropartphys.2011.01.006 arXiv:astro-ph.IM/1001.2834
[14] Cantarel B., Korf I., Robb SMC., Parra G., Ross E., Moore B., Holt C., Sanchez Alvarado A., and Yandell M. 2008. MAKER: An Easy-to-use Annotation Pipeline Designed for Emerging Model Organism Genomes. *Genome Research* 18, 1 (2008).
[15] B. A. Benson et al. 2014. SPT-3G: a next-generation cosmic microwave background polarization experiment on the South Pole telescope. In *Millimeter, Submillimeter, and Far-Infrared Detectors and Instrumentation for Astronomy VII (Proc. SPIE)*, Vol. 9153. Article 91531P, 91531P pages. https://doi.org/10.1117/12.2057305 arXiv:astro-ph.IM/1407.2973
[16] Ian Bird. 2011. Computing for the Large Hadron Collider. *Annual Review of Nuclear and Particle Science* 61, 1 (2011), 99–118.
[17] Peter Bui, Dinesh Rajan, Badi Abdul-Wahid, Jesus Izaguirre, and Douglas Thain. 2011. Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications. In *Workshop on Python for High Performance and Scientific Computing (PyHPC) at the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing)* .
[18] P Buncic, C Aguado Sanchez, J Blomer, L Franco, A Harutyunian, P Mato, and Y Yao. 2010. CernVM – a virtual software appliance for LHC applications. *Journal of Physics: Conference Series* 219, 4 (2010), 042003.
[19] J. Caballero et al. 2012. AutoPyFactory: A Scalable Flexible Pilot Factory Implementation. *Journal of Physics: Conference Series* 396 (2012), 032016.
[20] Scott Callaghan, Gideon Juve, Karan Vahi, Philip J. Maechling, Thomas H. Jordan, and Ewa Deelman. 2017. rvGAHP: Push-based Job Submission Using Reverse SSH Connections. In *Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science (WORKS '17).* ACM, New York, NY, USA, Article 3, 8 pages. https://doi.org/10.1145/3150994.3151003
[21] J. E. Carlstrom et al. 2011. The 10 Meter South Pole Telescope. *Proc. PASP* 123 (May 2011), 568. https://doi.org/10.1086/659879 arXiv:astro-ph.IM/0907.4445
[22] Ewa Deelman et al. 2015. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems* 46 (2015), 17–35. https://doi.org/10.1016/j.future.2014.10.008 Funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575.
[23] The CMS Collaboration (Adolphi R et al.). 2008. The CMS experiment at the CERN LHC. *Journal of Instrumentation* 3 (2008).
[24] Lisa Gerhardt, Wahid Bhimji, Shane Canon, Markus Fasel, Doug Jacobsen, Mustafa Mustafa, Jeff Porter, and Vakho Tsulaia. 2017. Shifter: Containers for HPC. *Journal of Physics: Conference Series* 898, 8 (2017), 082021. http://stacks.iop.org/1742-6596/898/i=8/a=082021
[25] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. 2017. Singularity: Scientific containers for mobility of compute. *PLOS ONE* 12, 5 (05 2017), 1–20. https://doi.org/10.1371/journal.pone.0177459
[26] Michael Litzkow, Miron Livny, and Matthew Mutka. 1988. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems.*
[27] Ruth Pordes et al. 2007. The open science grid. *Journal of Physics: Conference Series* 78, 1 (2007), 012057. http://stacks.iop.org/1742-6596/78/i=1/a=012057
[28] Reid Priedhorsky and Tim Randles. 2017. Charliecloud: Unprivileged Containers for User-defined Software Stacks in HPC. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17).* ACM, New York, NY, USA, Article 36, 10 pages. https://doi.org/10.1145/3126908.3126925
[29] I. Sfiligoi, D. C. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, and F. Wurthwein. 2009. The Pilot Way to Grid Resources Using glideinWMS. In *2009 WRI World Congress on Computer Science and Information Engineering*, Vol. 2. 428–432. https://doi.org/10.1109/CSIE.2009.950
[30] Douglas Thain and Miron Livny. 2005. Parrot: An Application Environment for Data-Intensive Computing. *Scalable Computing: Practice and Experience* 6, 3 (2005), 9–18.
[31] Douglas Thain, Todd Tannenbaum, and Miron Livny. 2003. Condor and the Grid. In *Grid Computing: Making the Global Infrastructure a Reality*, Fran Berman, Anthony Hey, and Geoffrey Fox (Eds.). John Wiley.
[32] Benjamin Tovar, Nicholas Hazekamp, Nathaniel Kremer-Herman, and Douglas Thain. 2018. Automatic Dependency Management for Scientific Applications on Clusters. In *IEEE International Conference on Cloud Engineering (IC2E)* .
[33] J. Towns et al. 2014. XSEDE: Accelerating Scientific Discovery. *Comp. in Sci. and Engr.* 16, 5 (Sept 2014), 62–74. https://doi.org/10.1109/MCSE.2014.80
[34] S. Tuecke, R. Ananthakrishnan, K. Chard, M. Lidman, B. McCollam, S. Rosen, and I. Foster. 2016. Globus auth: A research identity and access management platform. In *2016 IEEE 12th International Conference on e-Science (e-Science).* 203–212. https://doi.org/10.1109/eScience.2016.7870901
[35] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.