ARMY RESEARCH LABORATORY

VisIt Python-Based Job Launching

by Richard C. Angelini

ARL-TR-6493 **June 2013**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-6493 **June 2013**

VisIt Python-Based Job Launching

Richard C. Angelini
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
June 2013	Final	August 2012–March 2013

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
VisIt Python-Based Job Launching	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)	5d. PROJECT NUMBER
Richard C. Angelini	R.0006163.13
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
U.S. Army Research Laboratory ATTN: RDRL-CIH-S Aberdeen Proving Ground, MD 21005-5067	ARL-TR-6493

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

VisIt is a free interactive parallel visualization and graphical analysis tool that is commonly used at the U.S. Army Research Laboratory (ARL) for viewing scientific data on Linux, Windows, and OS X (Apple) workstations. An essential element for the successful implementation of VisIt and similar client-server application packages is the ability to connect a local client workstation to a remote high-performance computing system where the computed data resides. Prior to VisIt version 2.6.0, job launching was handled via a single perl script that was distributed as part of the VisIt. This script evolved over time to define the unique job-launching requirements for many of the systems supported by VisIt developers. However, after many years of use, this script became unwieldy and difficult to debug or modify because of the number of machine-specific "hacks" added to the code. VisIt 2.6.0 provided a new Python-based job-launching mechanism that was modular and extensible. This report describes the implementation of VisIt job launching at ARL.

15. SUBJECT TERMS

VisIt, python, HPC, job launching

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Richard C. Angelini
Unclassified	Unclassified	Unclassified	UU	26	19b. TELEPHONE NUMBER (*Include area code*) 410-278-6266

Contents

Acknowledgments

This work was supported in part by a grant of computer time and resources by the Department of Defense High-Performance Computing Modernization Program. I would like to thank Dr. Hank Childs of Lawrence Berkeley National Laboratory/University of California Davis for assistance and guidance in modifying the perl-based *internallauncher*, and Mr. Brad Whitlock of Lawrence Livermore National Laboratory for his assistance with implementing the Python-based *customlauncher* for VisIt 2.6.0 and for his overall assistance with the VisIt project.

1. Background

VisIt is a free interactive parallel visualization and graphical analysis tool commonly used at the U.S. Army Research Laboratory (ARL) for viewing scientific data on Linux, Windows, and OS X (Apple) workstations. Users can quickly generate visualizations from their data, animate their results through time, manipulate the viewpoint or geometric properties, and save the resulting images for presentations. VisIt contains a rich set of visualization features that allows data to be viewed in a variety of ways. VisIt can be used to visualize scalar and vector fields defined on two- and three-dimensional (2-D and 3-D) structured and unstructured meshes. VisIt was designed to handle very large dataset sizes in the terascale range and yet can also handle small datasets in the kilobyte range.[1]

VisIt was developed by the Department of Energy (DOE) Advanced Simulation and Computing Initiative (ASCI) to visualize and analyze the results of terascale simulations. It was developed as a framework for adding custom capabilities and rapidly deploying new visualization technologies. After an initial prototype effort, work on VisIt began in the summer of 2000, and the initial version of VisIt was released in the fall of 2002. Although the primary driving force behind the development of VisIt was for visualizing terascale data, it is also well suited for visualizing data from typical simulations on desktop systems.[1] VisIt has been in use at ARL for several years and is built from source code and distributed to numerous desktop workstations and high-performance computing (HPC) systems.

An essential element for the successful production implementation of VisIt and similar client-server application packages (EnSight, ParaView) is the ability to connect a local client workstation to a remote HPC system where the computed data resides. Today's modern low-cost Linux, Mac, and Windows desktop workstations with a standard commodity graphics card provide virtually any desktop system with sufficient power to drive these high-end visual analysis tools. The availability of these low-cost workstations combined with the availability of production-quality commercial (EnSight) and open-source (ParaView, VisIt) client-server visualization packages allows for unprecedented access to HPC-sized datasets from the desktop. These applications are client-server in nature; that is, there is a portion of the code (client-side) that runs on the local desktop workstation and is responsible for handling the graphical user interface (GUI)-based interface and the rendering and manipulation of the graphical components. On the HPC system, the server side of the application is responsible for the computationally intensive portions of the data postprocessing, such as reading in the simulation results, subsetting or manipulating the data, and applying advanced computational algorithms. The client and server

[1]VisIt Web site. About VisIt, https://wci.llnl.gov/codes/visit/about.html (accessed January 2013).

communicate with each other using standard network protocols; however, establishing a clear communication path between the client workstation and the allocated High-Performance Computing Modernization Program (HPCMP) resources is very challenging.[2]

These interactive client-server data analysis applications have been used for many years by more advanced computational researchers; however, use by the general population was limited by the complexities required to establish a clear communication path between the client and server. An interactive session required manually launching the client and server processes independently, establishing the appropriate SSH-tunnels to allow the processes to communicate, and then having it all come together into a working interactive session. These client-server applications had to be coerced to work within the HPC environment, overcoming obstacles such as the job queuing systems and any number of necessary security policies that restricted communication paths between the allocated back-end computational nodes and the client workstation.[2]

Developing automated methods for launching the HPC-side of the client-server connection is not a trivial process. Additionally, this task is inherently machine dependent, which makes the desire to develop a common set of launching tools nearly impossible. VisIt and other client-server applications therefore implement a framework of configuration tools that are customized to accommodate machine-specific job-launching details. In general, the launch sequence establishes a network connection to the HPC-side login node, submits a batch job through the local queuing system, and establishes communication from the allocated compute nodes back through the HPC login node and ultimately back to the client-side workstation. This job-launching framework relies on some form of a configuration file that resides on the client side to define the methodology required to establish the communication path to the desired host. The framework also relies on an application helper (shell, Python, or perl script) on the HPC login node to start the batch job and manage communication between the client workstation and the allocated compute nodes.

2. Implementation

Prior to VisIt version 2.6.0 (fall, 2012), job launching was handled via a single perl script named *internallauncher* that was distributed as part of the VisIt. This script evolved over time to define the unique job-launching requirements for many of the systems supported by VisIt developers. As new hosts were brought online, VisIt developers or code contributors modified this common perl script to add functionality required to support that new HPC system. The script included host information for DOE, the National Science Foundation, and university partners of VisIt. This information was not necessarily relevant to other computing sites. However, based on this all-inclusive single-script methodology, the burden of carrying around host information for every

[2]Angelini, R. Client-Server HPC Job Launching. *DOD HPC Insights*, Spring 2012, p 3.

machine supported by the VisIt developers was unavoidable. This common all-inclusive launch mechanism was actually beneficial at those HPC centers with systems supported by the VisIt developers as the software could be built, distributed, and put into use without any modifications from the local software support staff.

However, after many years of growth through the addition of new hosts and supported HPC centers, this script became unwieldy and difficult to debug or modify because of the number of machine-specific "hacks" added to the code. The task of adding a new host was particularly difficult for VisIt systems that were outside of those typically supported by the VisIt code developers, such as systems used by ARL and other Department of Defense (DOD) HPC systems. The ability of local support staff at those "outside" HPC centers to include additional host definitions required a significant amount of effort to determine not only which parameters need to be modified within the existing script, but where within the script those changes needed to be made. This was a challenging task made more difficult by the complexity of the changes to the script over time.

The *internallauncher* script sets machine-specific details such as environment variables and paths to important utilities, and includes "hacks" for more than 20 different HPC systems. A small subset of code that provides machine-specific customization within the *internallauncher* script released with the standard distribution of VisIt 2.5.0 looks like this:

```
# ----
# HACK for jaguarpf.ccs.ornl.gov
#
# ----
$IsRunningOnJaguar_ORNL = 0;
if (($parallel) and $exe_name eq "engine" and ($host =~ /jaguarpf/ or $host =~
/jaguar/))
{
    chomp($domain   = `hostname -d`);
    if ( $domain eq "ccs.ornl.gov" )
    {
        $IsRunningOnJaguar_ORNL = 1;
        $remotehost = $host;
        if ($host =~ /jaguarpf/)
        {
            $remotehost =~ s/jaguarpf-//;
        }
        else
        {
            $remotehost =~ s/jaguar/login/;
        }

        if ($ENV{PATH} eq "")
        {
            $ENV{PATH} = "/opt/torque/default/bin";
            $ENV{PATH} = join ':' , ("$ENV{PATH}","/usr/bin");
        }
        else
        {

            $ENV{PATH} = join ':' , ("$ENV{PATH}","/opt/torque/default/bin");
            $ENV{PATH} = join ':' , ("$ENV{PATH}","/usr/bin");
```

```
        }
    }
}
```

The *internallauncher* also includes the logic required to submit a job to the remote batch queuing system. The details of submitting a batch job are machine specific, and even a commonly used batch utility such as the Portable Batch System (PBS) has host-specific implementation details that increase the complexity of the *internallauncher* script. The script uses approximately 380 lines of code to handle just the PBS batch job utility qsub and includes code to support 6 different batch submission methods. A simple code example for the "salloc" batch job submission script looks like this:

```
# salloc

    elsif (substr($launch,0,6) eq "salloc")
    {
        @parcmd = ("salloc");
        push @parcmd, "-p", $part  if $part_set;
        push @parcmd, "-t", $time  if $time_set;
        push @parcmd, "-N", $nodes if $nodes_set;
        push @parcmd, "-n", $procs if $procs_set;
        if ($nodes_set)
        {
           $ppn = ceil($procs / $nodes);
           push @parcmd, "--ntasks-per-node=$ppn";
        }
        push @parcmd, "srun","-N1","-n1","--preserve-env","--mpi=none","mpirun";

        if ($nodes_set)
        {
           $ppn = ceil($procs / $nodes);
           push @parcmd, "--npernode", $ppn;
        }
        push @parcmd, @VisItcmd;
        if ($security_key_set) { push @parcmd, "-key", $security_key; }
        push @parcmd, @post_args;
        @printcmd = @parcmd;
        push @printcmd, ("\"".(pop @printcmd)."\"");
    }
```

In VisIt 2.5.0, the *internallauncher* included almost 4000 lines of perl code, with numerous deeply nested control flow statements required to accommodate the many hosts defined in this single launch file. For reasons previously described, the *internallauncher* script was quite difficult to modify and was a topic of numerous e-mail messages and personal discussions between the ARL principal investigator and members of the VisIt development team regarding the viability of this launch method. It was generally recognized by the VisIt development team that the *internallauncher* required an overhaul, and that it needed to become more modular and machine independent. Restructuring the job-launching methodology ultimately became an issue of allocating developer resources to implement a more modern and maintainable solution. The principal investigator responsible for supporting the client-server applications at ARL was in a unique position to provide input to the VisIt developers on the general direction of implementation of a new job launcher as there was extensive experience with job-launching

facilities in similar client-server software applications. Both EnSight and ParaView provided a framework from which automated job launching could be achieved; however, this framework was implemented in such a way as each host-specific job-launching script was independent, and the overall design was much more modular. Unlike VisIt, these applications did not rely on a common job-launching mechanism that contained configuration commands for multiple machines. The implementation in EnSight and ParaView, while unique for each package, provided a more maintainable method for job-launching implementation.

In the fall of 2012, VisIt developers were able to address the known issues with the job launching, and a new modular Python-based job-launching mechanism was introduced with VisIt 2.6.0. The new Python-based job launcher still relies on an *internallauncher* script as the underlying driver for client-server connectivity and batch job submission; however, the local site application specialist no longer needs to modify this script to make host-specific modifications. The updated *internallauncher* script does not contain any machine-specific details but establishes a default framework upon which local customizations can be based. A companion site-specific *customlauncher* script is created to include those unique details required to launch a job on a particular machine. By redefining Python methods that appear in the *internallauncher*, the *customlauncher* contains only those details required by the local site, and the complexity of supporting multiple HPC systems within the same script has been eliminated. The VisIt developers maintain the ability to provide job-launching mechanisms for HPC systems that they support by developing and distributing a unique *customlauncher* script for each of their supported systems. During software installation, the user has the ability to select one of these predefined host profiles, therefore maintaining the support for those systems specifically supported by the developers.

The new *internallauncher* script contains various Python classes that help launch VisIt commands:

- *JobSubmitter classes* let VisIt submit a parallel compute engine to a job control system.
- *Debugger classes* help launch VisIt under a debugger.
- *MainLauncher class* contains methods that are used to effect a launch.

The *internallauncher* function relies on the MainLauncher object "(or derived class)" to go through the various steps that are needed to run a VisIt program. The new launch system allows for a *customlauncher* file that contains a derived class of MainLauncher. The derived class can perform its own top-level specific initialization without polluting the main *internallauncher* script. Furthermore, since MPI launching is handled by various JobSubmitter classes, the derived MainLauncher class can return its own JobSubmitter classes that contain site-specific tweaks to MPI launching.[3]

[3]VisIt Users Web site. Visit Launcher. http://visitusers.org/index.php?title=VisIt_Launcher (accessed June 2013).

An example of using the *customlauncher* to redefine initialized values can be seen in this simple example. In the *internallauncher*, there is a method defined to determine how a particular flag is formatted to be included as part of a command line argument:

```
def SetupPPN(self, nodes, procs, ppn, use_vis):

   if use_vis:
       args = ["-l", "nodes=%s:ppn=%s:vis" % (nodes, ppn)]
   elif self.useppn:
       args = ["-l", "nodes=%s:ppn=%s" % (nodes, ppn)]
   else:
       args = ["-l", "nodes=%s" % nodes]
   return args
```

These values can be customized in the site-specific *customlauncher* using a derived class:

```
def SetupPPN(self, nodes, procs, ppn, use_vis):
# We could use nodes, procs, ppn to construct the arguments if a
# variable number of nodes or processors would be appropriate.

  args = ["-l", "place=scatter:excl","-l","select=%s:mpiprocs=%s" %(nodes,ppn)]

  if self.launcher.IsRunningOnHarold():
           args = ["-l", "place=scatter:excl", "-l", "select=
                                          %s:ncpus=8:mpiprocs=%s" % (nodes,ppn)]
  if self.launcher.IsRunningOnPitch():
            args = ["-l", "place=scatter:excl", "-l", "select=
                                         %s:ncpus=16:mpiprocs=%s" % (nodes,ppn)]
  return args
```

A functioning *customlauncher* example can be found in appendix A of this document. This file, when placed in the same directory as the unedited *internallauncher* script, automatically includes those local customizations required to define the job-launching parameters for a specific host when VisIt is launched. In addition, appendix B includes a sample of a VisIt host profile that is created through the GUI interface and can be saved out in an Extensible Markup Language (XML) format. The host profile provides specific details required to define the parameters required to connect to a remote server system and allows for the definition of details required to submit a batch job to the queuing system on that remote system. Once defined, a host profile can be shared within the VisIt software distribution so that a particular machine definition can be included and shared among the user community.

3. Conclusion

The Python-based job-launching solution implemented in VisIt 2.6.0 eliminates many of the issues associated with the perl-based *internallauncher* script by creating a modular framework for local customizations. This new implementation allows an experienced application Python programmer to extend and customize the launch methods in a modular way, independent of the

supplied *internallauncher* script. Unlike the perl-based solution, there is no confusion associated with supporting all of the known HPC systems in a single, complicated command file.

Creation of the Python-based *customlauncher* requires an extensive working knowledge of Python and a thorough understanding of how the existing *internallauncher* works. The ability to use derived Python classes or to modify predefined Python methods is not necessarily an intuitive process. Generating the code modifications required to support local systems without reviewing *customlauncher* examples developed at other locations, or an extensive amount of assistance from VisIt developers, is intimidating for the casual Python code developer or for someone not familiar with the general philosophy of VisIt job launching. However, once the general framework for a locally developed *customlauncher* is developed, it is possible to leverage that code for use in the support of additional HPC systems.

The implementation of the Python-based *internallauncher* script made available in VisIt 2.6.0 is a significant improvement over the perl-based all-inclusive job-launching script. The modularity and extensibility of the framework implemented in the updated *internallauncher* provides a basis to develop a maintainable application code. These improvements are a step in the right direction, and hopefully future releases of the VisIt will focus attention on reducing the complexity of developing a site-specific *customlauncher* script. In keeping with a machine-independent solution, perhaps a directives-based XML approach can be considered an adjunct to the *customlauncher* script. The ability to easily develop and implement a job-launching mechanism is essential for subject matter experts to provide site-specific application support. As it stands, however, the current Python-based implementation has greatly improved this ability.

INTENTIONALLY LEFT BLANK.

Appendix A. Sample *customlauncher* Script

This *customlaunch* script defines host-specific launching parameters for high-performance computing systems located at ARL, including MJM, Harold, Pitch and Pershing.

```
################################################################################
# Class: JobSubmitter_qsub_ARL
#
# Purpose:    Custom qsub launcher for ARL & DOD HPCMP
#
# Programmer: Rick Angelini
#
# Modifications:
#
################################################################################

class JobSubmitter_qsub_ARL(JobSubmitter_qsub):
    def __init__(self, launcher):
        super(JobSubmitter_qsub_ARL, self).__init__(launcher)

    def TFileLoadModules (self, tfile):
        if self.launcher.IsRunningOnPitch():
                print "IsRunningOnPitch: Adding modules to tfile"
                tfile.write("eval `modulecmd sh purge`\n")
                tfile.write("eval `modulecmd sh load pbs Master`\n")
                tfile.write("eval `modulecmd sh load compiler/gcc/4.4`\n")
                tfile.write("eval `modulecmd sh load mpi/openmpi/1.6.0`\n")
                tfile.write("eval `modulecmd sh list` \n")
                tfile.write("cat $PBS_NODEFILE\n")

        if self.launcher.IsRunningOnPershing():
                print "IsRunningOnPershing: Adding modules to tfile"
                tfile.write("eval `modulecmd sh purge`\n")
                tfile.write("eval `modulecmd sh load pbs Master`\n")
                tfile.write("eval `modulecmd sh load compiler/gcc/4.4`\n")
                tfile.write("eval `modulecmd sh load mpi/openmpi/1.6.0`\n")
                tfile.write("eval `modulecmd sh list` \n")
                tfile.write("cat $PBS_NODEFILE\n")

        if self.launcher.IsRunningOnUtill():
                print "IsRunningOnUtilityServer: Adding modules to tfile"
                tfile.write("source /app/modules/init/sh\n")
                tfile.write("module switch compiler compiler/gcc/4.1\n")
                tfile.write("module switch mpi mpi/gnu/openmpi/1.4.3\n")
                tfile.write("module list\n")
                tfile.write("cat $PBS_NODEFILE\n")

        if self.launcher.IsRunningOnHarold():
                print "IsRunningOnHarold: Adding modules to tfile"
                tfile.write("eval `modulecmd sh purge`\n")
                tfile.write("eval `modulecmd sh load modules pbs Master`\n")
                tfile.write("eval `modulecmd sh load visit/2.6.0`\n")
                tfile.write("eval `modulecmd sh list`\n")
                tfile.write("cat $PBS_NODEFILE\n")

    def mpirun_args(self, args):
      # Change mpi launch command on MJM
      if self.launcher.IsRunningOnMJM():
            mpicmd = ["openmpirun.pbs"]
          mpicmd = mpicmd + ["-np", self.parallel.np]
      else:
```

```
          mpicmd = ["mpirun"]
          mpicmd = mpicmd + ["-np", self.parallel.np]

        if self.parallel.sublaunchargs != None:
            mpicmd = mpicmd + self.parallel.sublaunchargs
        if self.parallel.machinefile != None:
            mpicmd = mpicmd + ["-machinefile", self.parallel.machinefile]
        mpicmd = mpicmd + self.VisItExecutable()
        mpicmd = mpicmd + ["-plugindir", GETENV("VISITPLUGINDIR")]
        mpicmd = mpicmd + args

        # Return the mpicmd list
        return mpicmd

    def SetupPPN(self, nodes, procs, ppn, use_vis):
        # We could use nodes, procs, ppn to construct the arguments if a
        # variable number of nodes or processors would be appropriate.
        args = ["-l", "place=scatter:excl", "-l", "select=%s:mpiprocs=%s" %
                                                                (nodes,ppn) ]

      if self.launcher.IsRunningOnHarold():
            args = ["-l", "place=scatter:excl", "-l", "select=
                                        %s:ncpus=8:mpiprocs=%s" % (nodes,ppn) ]

      if self.launcher.IsRunningOnPitch() or
                                          self.launcher.IsRunningOnPershing():
                args = ["-l", "place=scatter:excl", "-l", "select=
                                       %s:ncpus=16:mpiprocs=%s" % (nodes,ppn) ]

        return args


##############################################################################
# Class: ARLLauncher
#
# Purpose:    Custom launcher for ARL & DoD HPCMP Systems
#
# Programmer: Rick Angelini
#
# Modifications:
#
##############################################################################

class ARLLauncher(MainLauncher):
    def __init__(self):
        super(ARLLauncher, self).__init__()

      self.pitch = -1
      self.pershing = -1
      self.utill = -1
      self.harold = -1
      self.mjm = -1

    def IsRunningOnPitch(self):
        if self.pitch == -1:
                self.pitch = 0
                if self.parallelArgs.parallel and \
                        self.generalArgs.exe_name == "engine" and \
                        (self.sectorname() == "pitch-login" or self.sectorname()
                                                              == "pitch" or \
                        self.sectorname() == "pitch-l"):
                                print "I AM ON PITCH"
                                self.pitch=1
                        self.generalArgs.host = self.nodename() + "-ib"
```

```
                        self.generalArgs.host = "pitch-login1-ib"  #HACK
#                       print "Changing self.generalArgs.host=" +
                                                            self.generalArgs.host
        return self.pitch

    def IsRunningOnPershing(self):
        if self.pershing == -1:
                self.pershing = 0
                if self.parallelArgs.parallel and \
                        self.generalArgs.exe_name == "engine" and \
                        (self.sectorname() == "pershing-login" or
                                        self.sectorname() == "pershing" or \
                        self.sectorname() == "pershing-l"):
                                print "I AM ON PERSHING"
                                self.pershing=1
                        self.generalArgs.host = self.nodename() + "-ib"
#                       print "Changing self.generalArgs.host=" +
self.generalArgs.host
        return self.pershing

    def IsRunningOnUtill(self):
         if self.utill == -1:
                self.utill = 0
                if self.parallelArgs.parallel and \
                        self.generalArgs.exe_name == "engine" and \
                        self.sectorname()[2:] == "utill-":
                                print "I AM on a UTILITY SERVER: "  +
                                                              self.nodename()
                                self.utill = 1
         return self.utill

    def IsRunningOnHarold(self):
         if self.harold == -1:
                self.harold = 0
                if self.parallelArgs.parallel and \
                        self.generalArgs.exe_name == "engine" and \
                        self.sectorname() == "harold-l":
                                print "I AM ON Harold: " + self.nodename()
                                self.harold = 1
         return self.harold

    def IsRunningOnMJM(self):
         if self.mjm == -1:
                self.mjm = 0
                if self.parallelArgs.parallel and \
                        (self.generalArgs.exe_name == "vcl" or
self.generalArgs.exe_name == "engine") and \
                        self.sectorname() == "l":
                                print "I AM ON MJM: " + self.nodename()
                                self.mjm = 1
         return self.mjm

    def Customize(self):
        # ----
        # Pitch @ ARL
        # ----
      if self.IsRunningOnPitch ():
            paths = self.splitpaths(GETENV("LD_LIBRARY_PATH"))
            addedpaths = ["/usr/cta/unsupported/openmpi/gcc/4.4.0/openmpi-
                                                    1.6/lib:/opt/pbs/default/lib"]
            SETENV("LD_LIBRARY_PATH", self.joinpaths(paths + addedpaths))

            paths = self.splitpaths(GETENV("PATH"))
            addedpaths = ["/usr/cta/unsupported/openmpi/gcc/4.4.0/openmpi-1.6/bin"]
```

```
            SETENV("PATH", self.joinpaths(paths + addedpaths))

          # ----
          # Pershing @ ARL
          # ----
        if self.IsRunningOnPershing ():
              paths = self.splitpaths(GETENV("LD_LIBRARY_PATH"))
              addedpaths = ["/usr/cta/unsupported/openmpi/gcc/4.4.0/openmpi-
                                                    1.6/lib:/opt/pbs/default/lib"]
              SETENV("LD_LIBRARY_PATH", self.joinpaths(paths + addedpaths))

              paths = self.splitpaths(GETENV("PATH"))
              addedpaths = ["/usr/cta/unsupported/openmpi/gcc/4.4.0/openmpi-1.6/bin"]
              SETENV("PATH", self.joinpaths(paths + addedpaths))

          # ----
          # All DSRC Utility Servers
          # ----
        if self.IsRunningOnUtill ():
              paths = self.splitpaths(GETENV("LD_LIBRARY_PATH"))
              addedpaths = ["/app/openmpi/gnu/1.4.3/lib:/usr/lib64"]
              SETENV("LD_LIBRARY_PATH", self.joinpaths(paths + addedpaths))

              paths = self.splitpaths(GETENV("PATH"))
              addedpaths = ["/app/cwjm/20110609/bin"]
              SETENV("PATH", self.joinpaths(paths + addedpaths))

          # ----
          # Harold @ ARL
          # ----
        if self.IsRunningOnHarold ():
              paths = self.splitpaths(GETENV("LD_LIBRARY_PATH"))
              addedpaths = ["/usr/cta/unsupported/openmpi/gcc/4.1/openmpi-1.4.1/lib"]
              SETENV("LD_LIBRARY_PATH", self.joinpaths(paths + addedpaths))

              paths = self.splitpaths(GETENV("PATH"))
              addedpaths = ["/usr/cta/unsupported/openmpi/gcc/4.1/openmpi-1.4.1/bin"]
              SETENV("PATH", self.joinpaths(paths + addedpaths))

          # ----
          # MJM @ ARL
          # ----
        if self.IsRunningOnMJM ():
              paths = self.splitpaths(GETENV("LD_LIBRARY_PATH"))
              addedpaths =
["/opt/compiler/gcc/4.4/lib64:/opt/compiler/gcc/4.4/lib:/opt/mpi/x86_64/gcc/4.4/
openmpi-1.3/lib"]
              SETENV("LD_LIBRARY_PATH", self.joinpaths(paths + addedpaths))

              paths = self.splitpaths(GETENV("PATH"))
              addedpaths = ["/opt/mpi/x86_64/gcc/4.4/openmpi-1.4/bin"]
              SETENV("PATH", self.joinpaths(paths + addedpaths))

    #
    # Override the JobSubmitterFactory method so the custom job submitter can
    # be returned.
    #
    def JobSubmitterFactory(self, launch):
        if launch[:4] == "qsub" or launch[:4] == "msub":
            return JobSubmitter_qsub_ARL(self)
        return super(ARLLauncher, self).JobSubmitterFactory(launch)


    # DAAC LOGGING
```

```
    #
    # Determine when we're doing server side logging.
    def ServerSideLogging(self):
      print "self.generalArgs.exe_name=" + self.generalArgs.exe_name
        comp = self.generalArgs.exe_name in ["engine", "engine_ser",
"engine_par"]
      print "comp=" + str(comp)
      return comp

    # Override the Logging() method. This method gets called from self.call when
    # we launch a program and we're doing logging.
    def Logging(self, args):
        logger_cmd = []
        self.logging=1
        if self.logging:
            if self.ServerSideLogging():
            print "ServerSideLogging discovered"
                short_host = self.nodename()
                nodes = "0"
                if self.parallelArgs.nn != None:
                    nodes = self.parallelArgs.nn
                procs = "0"
                if self.parallelArgs.np != None:
                    procs = self.parallelArgs.np
                time = "0"
                if self.parallelArgs.time != None:
                    time = self.parallelArgs.time

                # Only log server-side usage if it's on a defined HPC resource
              if self.IsRunningOnPitch () or \
               self.IsRunningOnPershing () or \
               self.IsRunningOnUtill () or \
             self.IsRunningOnHarold () or \
               self.IsRunningOnMJM ():
                      logger_cmd = ["%s%s" %
          (os.path.dirname(GETENV("VISITHOME")),"/utils/daac_logger"), "remote",
                 "visit_server", self.visitver, short_host, nodes, procs, time]
                      os.system(" ".join(str(x) for x in logger_cmd))

            elif self.generalArgs.exe_name == "viewer":
                logger_cmd = ["%s%s" % (os.path.dirname(GETENV("VISITHOME")),
                   "/utils/daac_logger"), "local", "LINUX_Visit", self.visitver]
            os.system(" ".join(str(x) for x in logger_cmd))

# Launcher creation function
def createlauncher():
    return ARLLauncher()
```

Appendix B. Sample VisIt Host Profile Configuration File

The host profile provides specific details required to define the parameters required to connect to a remote server system and allows for the definition of details required to submit a batch job to the queuing system on that remote system. This particular example defines a serial connection that would run on a login node on the remote system, while the parallel machine profile submits a parallel job through the queuing system.

```
<?xml version="1.0"?>
<Object name="MachineProfile">
    <Field name="hostNickname" type="string">HAROLD</Field>
    <Field name="host" type="string">harold.xx.xx.mil</Field>
    <Field name="userName" type="string">notset</Field>
    <Field name="hostAliases" type="string">harold Harold HAROLD</Field>
    <Field name="directory" type="string">/usr/local/visit</Field>
    <Field name="shareOneBatchJob" type="bool">false</Field>
    <Field name="sshPortSpecified" type="bool">false</Field>
    <Field name="sshPort" type="int">0</Field>
    <Field name="clientHostDetermination" type="string">MachineName</Field>
    <Field name="manualClientHostName" type="string"></Field>
    <Field name="tunnelSSH" type="bool">true</Field>
    <Object name="LaunchProfile">
        <Field name="timeout" type="int">240</Field>
        <Field name="numProcessors" type="int">2</Field>
        <Field name="numNodesSet" type="bool">true</Field>
        <Field name="numNodes" type="int">2</Field>
        <Field name="partitionSet" type="bool">true</Field>
        <Field name="partition" type="string">debug</Field>
        <Field name="bankSet" type="bool">true</Field>
        <Field name="bank" type="string"></Field>
        <Field name="timeLimitSet" type="bool">true</Field>
        <Field name="timeLimit" type="string">00:15:00</Field>
        <Field name="launchMethodSet" type="bool">true</Field>
        <Field name="launchMethod" type="string">qsub/mpirun</Field>
        <Field name="forceStatic" type="bool">true</Field>
        <Field name="forceDynamic" type="bool">false</Field>
        <Field name="active" type="bool">false</Field>
        <Field name="arguments" type="stringVector"></Field>
        <Field name="parallel" type="bool">true</Field>
        <Field name="launchArgsSet" type="bool">true</Field>
        <Field name="launchArgs" type="string">"-l application=visit -N
VisIt"</Field>
        <Field name="sublaunchArgsSet" type="bool">false</Field>
        <Field name="sublaunchArgs" type="string"></Field>
        <Field name="sublaunchPreCmdSet" type="bool">false</Field>
        <Field name="sublaunchPreCmd" type="string"></Field>
        <Field name="sublaunchPostCmdSet" type="bool">false</Field>
        <Field name="sublaunchPostCmd" type="string"></Field>
        <Field name="machinefileSet" type="bool">false</Field>
        <Field name="machinefile" type="string"></Field>
        <Field name="visitSetsUpEnv" type="bool">false</Field>
        <Field name="canDoHWAccel" type="bool">false</Field>
        <Field name="havePreCommand" type="bool">false</Field>
        <Field name="hwAccelPreCommand" type="string"></Field>
        <Field name="havePostCommand" type="bool">false</Field>
        <Field name="hwAccelPostCommand" type="string"></Field>
        <Field name="profileName" type="string">HAROLD Parallel</Field>
    </Object>
```

```
    <Object name="LaunchProfile">
        <Field name="timeout" type="int">240</Field>
        <Field name="numProcessors" type="int">2</Field>
        <Field name="numNodesSet" type="bool">true</Field>
        <Field name="numNodes" type="int">2</Field>
        <Field name="partitionSet" type="bool">true</Field>
        <Field name="partition" type="string">debug</Field>
        <Field name="bankSet" type="bool">true</Field>
        <Field name="bank" type="string"></Field>
        <Field name="timeLimitSet" type="bool">true</Field>
        <Field name="timeLimit" type="string">00:15:00</Field>
        <Field name="launchMethodSet" type="bool">true</Field>
        <Field name="launchMethod" type="string">qsub/mpirun</Field>
        <Field name="forceStatic" type="bool">true</Field>
        <Field name="forceDynamic" type="bool">false</Field>
        <Field name="active" type="bool">false</Field>
        <Field name="arguments" type="stringVector"></Field>
        <Field name="parallel" type="bool">false</Field>
        <Field name="launchArgsSet" type="bool">true</Field>
        <Field name="launchArgs" type="string">"-l application=visit"</Field>
        <Field name="sublaunchArgsSet" type="bool">false</Field>
        <Field name="sublaunchArgs" type="string"></Field>
        <Field name="sublaunchPreCmdSet" type="bool">false</Field>
        <Field name="sublaunchPreCmd" type="string"></Field>
        <Field name="sublaunchPostCmdSet" type="bool">false</Field>
        <Field name="sublaunchPostCmd" type="string"></Field>
        <Field name="machinefileSet" type="bool">false</Field>
        <Field name="machinefile" type="string"></Field>
        <Field name="visitSetsUpEnv" type="bool">false</Field>
        <Field name="canDoHWAccel" type="bool">false</Field>
        <Field name="havePreCommand" type="bool">false</Field>
        <Field name="hwAccelPreCommand" type="string"></Field>
        <Field name="havePostCommand" type="bool">false</Field>
        <Field name="hwAccelPostCommand" type="string"></Field>
        <Field name="profileName" type="string">HAROLD Serial</Field>
    </Object>
    <Field name="activeProfile" type="int">0</Field>
</Object>
```

INTENTIONALLY LEFT BLANK.

Bibliography

Angelini, R. EnSight HPC Job Launching. *DOD HPC Insights*, Spring 2011, p 10.

Hand, R. ParaView Client-Server on Crays at the ERDC DSRC. *DOD HPC Insights*, Spring 2010, p 17.

Martin, J.; Angelini, R.; Vickery, R. Simplified SSH Tunnels for ParaView Client/Server. *DOD HPC Insights*, Spring 2009, p 27.

NO. OF
COPIES ORGANIZATION

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

1 (PDF) DIRECTOR US ARMY RESEARCH LAB IMAL HRA

1 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL

1 (PDF) GOVT PRINTG OFC A MALHOTRA

ABERDEEN PROVING GROUND

1 (PDF) DIR USARL RDRL CIH S R C ANGELINI