Bladed Linux
Batch Processing

DISCLAIMER

Acceptance of this document by the client is on the basis that Garrad Hassan & Partners Ltd is not in any way to be held responsible for the application or use made of the findings of the results from the analysis and that such responsibility remains with the client.

COPYRIGHT

All rights reserved. Duplications of this document in any form are not allowed unless agreed in writing by Garrad Hassan & Partners Ltd.

© 2020 Garrad Hassan & Partners Ltd.

DNV GL

One Linear Park, Avon Street, Temple Quay, Bristol, BS2 0PS, UK

www.dnvgl.com

bladed@dnvgl.com

Table of Contents

1 BACKGROUND 3

2 AIM 3

3 SCOPE 3

4 LIMITATIONS 3

5 BLADED ON LINUX 3
5.1 System Requirements 3
5.2 Unsupported Functionality 4

6 BLADED BATCH PROCESSING 4
6.1 On Windows 4
6.2 Transitioning to Linux 4
6.2.1 Input Preparation for Linux 6

7 LINUX BATCH PROCESSING TOOLS 6

8 HTCONDOR – BASICS 9
8.1 The Central Manager 9
8.2 Execute Nodes 10
8.3 Submit Nodes 10
8.4 HTCondor Daemons 10

9 HTCONDOR – SETTING UP 11
9.1 Pre-requisites 11
9.2 Preparation 11
9.3 Setting Up A Test Pool 12
9.3.1 Prepare the nodes for condor installation 12
9.3.2 Install HTCondor on both machines 12
9.3.3 Configure the HTCondor Installation 13
9.3.4 Start HTCondor 15
9.4 Installing and Configuring Bladed 15
9.5 Creating a shared file system 18
9.6 Creating Sample Runs 19
9.7 Submitting jobs to the pool 21
9.8 Scaling Up the Pool 22

10 NEXT STEPS 24

1 BACKGROUND

The main Bladed calculation engine - dtbladed, has been available on the Linux platform from version 4.10 of Bladed. The Linux version is expected to deliver significant reduction in compute costs for high throughput scenarios due to the relatively lower cost of Linux compute nodes – both locally and on third party Cloud platforms. However, most Bladed users have so far been unable to realise this cost reduction due to the absence of a reliable batch processing system around Bladed for Linux. The existing batch program for Bladed uses Windows specific technology and cannot manage simulations on the Linux platform. However, many third-party solutions exist in this space and offer a quick route to setting up highly parallelised Bladed workflows on Linux.

2 AIM

This document aims to provide information and instructions for setting up a batch processing system built around the Linux version of Bladed using third-party tools. The resulting setup is expected to require extension and further configuration before it can be production ready. This guide does not set out to be exhaustive or definitive in how to set up a chosen third-party tool but will hopefully help lay the foundations for a reliable and performant solution.

3 SCOPE

The target audience for this document are Bladed users/tools developers who are tasked with setting up a high throughput compute infrastructure and are already familiar with Bladed as a tool.

Bladed is used in a wide variety of custom workflows within enterprises, so it is impractical to provide instructions at a level of detail that will address all these scenarios. This guide will therefore focus on aspects of batch processing that are universally applicable.

4 LIMITATIONS

Recommendations made in this guide are based on the most typical batch processing scenario for Bladed, where the user has a custom process for generating inputs for their simulations (i.e. runs) and uses a dedicated set of compute nodes for parallelising the simulations. Inputs and outputs are assumed to be stored on a shared file system, accessible from all the nodes in the compute infrastructure for reading and writing.

Out of scope: This guide does not make recommendations or provide instructions on managing data security and implementing access control for multiple users. Instructions in the guide will produce an "all permissive" setup which will need to be secured as per specific IT/system requirements at the users end.

5 BLADED ON LINUX

5.1 System Requirements

Operating System	Linux (Ubuntu 18.04 LTS – Bionic Beaver)
Processor	2.5 GHz
SIMD Instruction Set	SSE2
RAM	4 GB
Storage	2 GB, preferably SSD
Licencing	A valid Bladed network licence with the

	"processor only" feature set

5.2 Unsupported Functionality

On Linux, only the dtbladed component is provided and therefore users cannot complete the following tasks:

- Generation of wind files
- Post-processing of calculation results
- Generation of quake
- Pre-processing calculations
- Bladed Hardware Test Module
- Use of the Bladed GUI

Users should continue to use their Windows Bladed installations to perform these tasks. The output produced by the Linux version of the calculation uses the same format as the Windows version, so it is possible to seamlessly link Linux calculations with your existing Windows post-processing workflow.

6 BLADED BATCH PROCESSING

Setting up and performing large numbers of simulations is a common task in nearly all Bladed workflows. The Bladed calculation engine is a single threaded executable that is predominantly CPU bound and individual calculations range from a few minutes at the lower end to multiple hours at the upper end and often produces many hundreds of megabytes of output data per calculation. The primary bottlenecks in most Bladed workflows are therefore compute capacity, followed by data storage capacity and network latency.

Batch processing allows users to increase throughput of workloads by spreading the runs across as many compute nodes as possible, while still providing a consistent runtime and file storage access across all nodes. Note that the primary goal here is not to run individual calculations as quickly as can be (i.e. low latency), but to get the overall workload to finish the quickest (i.e. high throughput) with an acceptable latency.

6.1 On Windows

Bladed Batch is a standalone module of Bladed that provides batch processing capability for Windows users. It is fully integrated with the Bladed desktop application and can manage tens of compute nodes within a local network and shared work among them. The tight integration with Bladed allows this module to support application specific operations which would be absent in an off the shelf batch processing solution. While the local processing capacity manageable through Bladed Batch is limited, it provides a powerful Cloud backend for those users that need higher throughput.

This module however is limited to the Windows platform due to its reliance on Windows specific technologies and communication protocols.

6.2 Transitioning to Linux

Transitioning from a Windows native workflow for Bladed to one where the Linux version of the calculation is used involves introducing some significant changes. Diagram below illustrates a simplified Bladed workflow which will form the basis for this discussion.

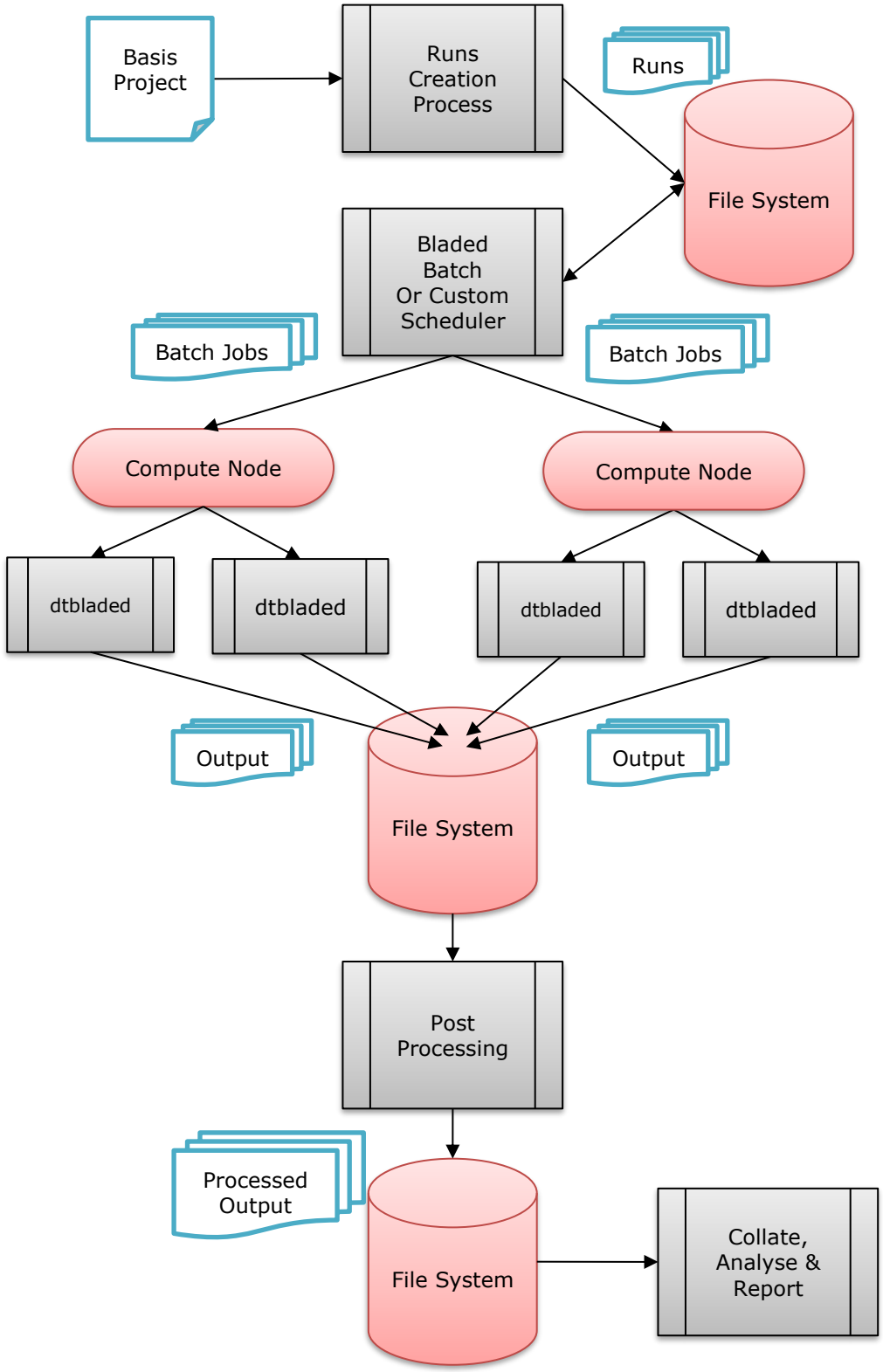


Figure 1: Simplified Bladed Workflow

The simplified workflow above can be understood in terms of four distinct phases:

1. Preparing inputs – Creation of inputs for runs from a basis project based on some pre-defined scheme of variation (e.g. IEC standards)
2. Performing simulations – The compute intensive operation where the previously created runs are distributed among multiple machines with the help of a batch processing system. Results from these simulations are written back to a shared file system.
3. Post Processing – Output from simulations are post-processed using dtsignal (the Bladed post processing tool) or custom implementations of post processing algorithms.
4. Analysis and reporting – Collation, analysis and reporting of post processed results.

The transition under discussion relates primarily to phases 1 and 2 of this process.

6.2.1 Input Preparation for Linux

The primary input to the dtbladed calculation is a text-based input file (DTBLADED.IN) which captures the model under test, environment conditions, operating conditions/faults and simulation configuration. In order to run on Linux, all Windows specific entries in this file should be identified and transformed to their Linux equivalent. In most cases, this simply *involves translating Windows path structures (absolute and relative) to paths that will resolve to a valid location on the Linux compute nodes* where the simulations will now be performed. Note that Linux also uses "/" as the directory separator character instead of "\" which is the default separator on Windows although Windows also allows use of the "/" character.

Common examples of such values include the "PATH" value which represent the output location for the run, location of other inputs such as turbulent wind files, load time-histories and other ancillary files such as external controller dll, pitch dll, external loads dll as well as parameter files for these components.

Depending on whether you would like to move all compute to Linux or retain the flexibility to mix your compute resources, you may choose to perform this translation right at the point of input creation or just before a run is requested. Use of shared file systems that are mounted to pre-defined target locations on the Linux compute nodes will allow you to perform such translations for paths easily. For example, if an input file path on Windows can be represented as X:\Project_Folder\Wind_Files\10mps_seed12.wnd, you could mount this shared file system at /mnt/Project_Folder thereby getting a direct translation from X:\ to /mnt/. Ancillary inputs and custom dlls

In addition to translating Windows paths to their equivalent Linux paths, you will also need to convert any custom dynamic link library (dll) that you use within your simulation to their Linux equivalent. For example, you will typically need to recompile your external controller to a Linux shared object file (.so). While doing this you will need to make sure that the recompilation targets the C++ 11 language standard and runtime version libstdc++6 as supported by Bladed Linux. In most cases, this task is expected to be straightforward assuming you have access to the source code and have some familiarity with Linux tools for the language in which it is written. DNV GL may be able to offer guidance and some assistance in this task. Please contact Renewables.support@dnvgl.com.

7 LINUX BATCH PROCESSING TOOLS

Most Linux operating systems come with built in tools for the user to create a crude batching system around. Commands such as "at", "atq", "atrm", "batch" and the crontab mechanism form this core of features. However, most real-world use cases will very soon grow in scope beyond what is achievable through these basic tools. Hence it is recommended that users looking to utilizing Linux for high

throughput Bladed workloads should start by looking at third party tools rather than attempting to build their own.

There is a wide array of new and relatively old third-party systems and tools that address the question of Batch processing on Linux. Some examples include:

- HTCondor
- Microsoft HPC Pack (Windows based, but supports Linux nodes)
- Celery
- RQ (RedisQueue)
- Dask
- Huey

Each of these tools come with their own set of unique advantages and disadvantages when applied in the context of a Bladed workflow. However, they can be subdivided into two broad categories; tools that are designed for parallelising compute heavy processes such as Bladed simulations and tools that are primarily designed for generic out of band processing which can be adapted for use in the Bladed context. For the purpose of this paper, we evaluated two tools - HTCondor and Celery, which represent these two categories respectively; these are not recommendations or endorsements but a suitability study around a typical Bladed use case. All 3rd party supplier summary information is provided in good faith and is not intended to be definitive / exhaustive.

	Celery	**HTCondor**
Primary design goal	Out of band processing of compute requests.	High Throughput Computing (HTC) on large collections of distributed computing resources.
System summary	Central message broker with distributed workers and a global results store. Broker and results store functions are provided by commercial tools.	Collection of daemons deployed on multiple nodes forming a distributed network.
Owner/maintainer	celeryproject.org	Centre for HTC at University of Wisconsin-Madison, USA
Platform support	Cross-platform	Cross-platform
Licence	Celery – Open source, BSD Licence Broker/Results Store – Licence based on choice of component	Open source, Apache Licence Version 2.0 HTCondor Licence Notes
In active dev?	Yes	Yes
Paid support	No	Yes
Primary Languages	Python	C++, Python

Primary interface	Command line (python terminal)	Command line
Graphical user interface	Available through the "Flower" add-on.	CondorGUI and JCondor – both from third parties. Limited functionality for job monitoring and management.
Scalability	No defined limit. Hard to find real-world figures.	Extremely high. Real world examples of 40K parallel slots.
Support for hybrid pools (mixing hardware, on-prem and in Cloud)	Yes	Yes
Variants	None	HTCondor – multi-user MiniCondor – single node installation for one user
Can be containerised	Yes	Yes
Security and user management	Patchy	Robust and fine grained
API present	Yes (Web)	Yes (Web)
Cycle scavenging support	No	Yes
Task chaining	Yes – Group, chain and chord	Yes – support for Directed Acyclic Graph (DAG) based task chaining
Input/output file management	No	Yes
Checkpointing and migration of jobs	No	Yes (Not applicable to dtbladed calculations although some post processing calcs may benefit)

In our experience of the tools, while Celery is easier to get started with, the learning curve gets drastically steeper as you attempt to tackle real world use cases for Bladed. HTCondor, on the other hand, can seem harder to get going with, but impresses with its rich set of functionality and reliable performance.

Having performed a detailed comparison and having set up sample clusters using both candidate technologies, we have concluded that HTCondor better fits the typical usage context around Bladed. It is a very mature product with a proven track record and a highly refined set of functionalities geared towards distributed scientific computing. However, we would strongly urge users to perform their own analysis and requirement matching before embarking on their Linux batch processing journey.

8 HTCONDOR – BASICS

The core of the HTCondor system design can be represented as below:

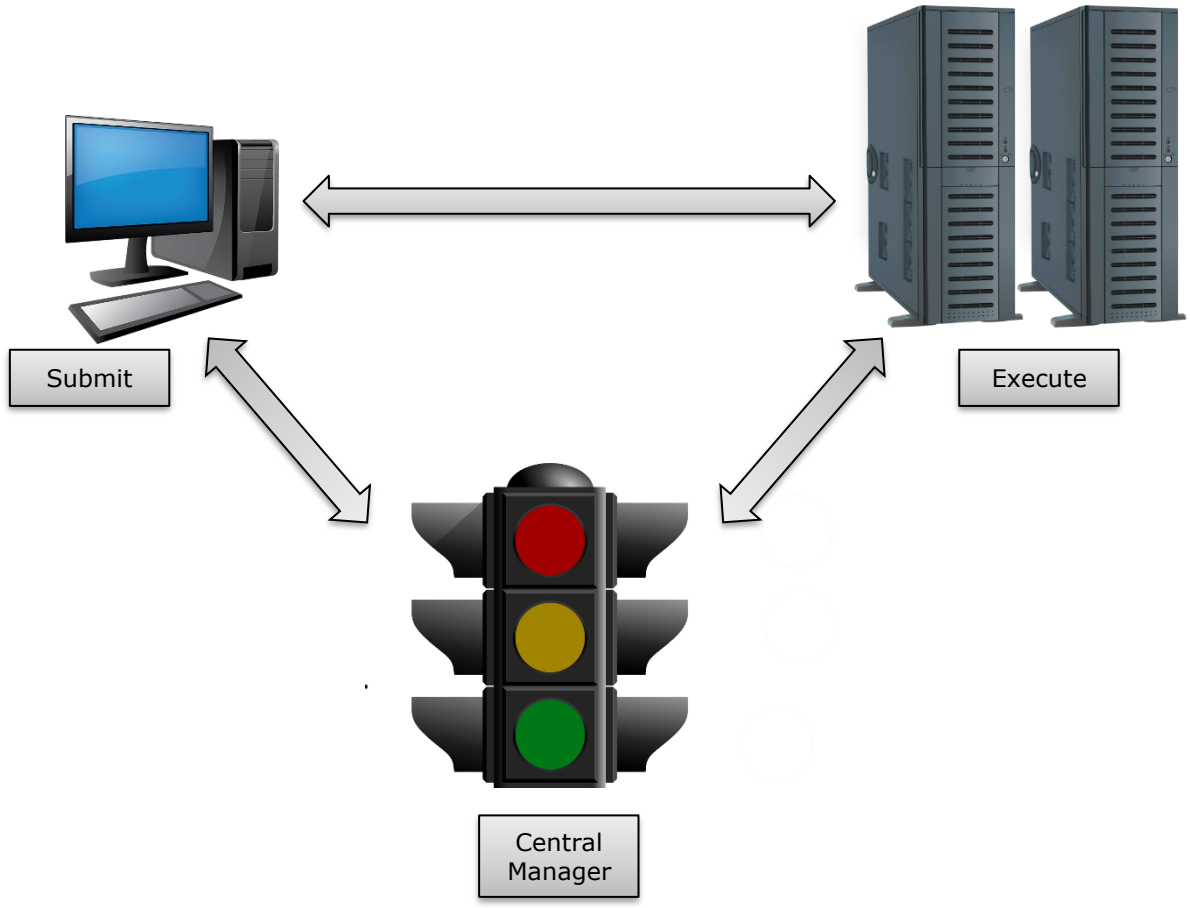


Figure 2: HTCondor System

The system is composed of services that provide functionality that could be grouped into one of three node types - submit, execute and central manager. Submit nodes queue up the jobs for execute nodes to perform and the central manager orchestrates the process. Jobs are submitted with a definition of requirement that should be met by an execute node that can run them. Execute nodes on the other hand define the resources and capabilities they have on offer. A match-making algorithm that has view of both jobs and available resources then tries to find an appropriate slot for each submitted job.

8.1 The Central Manager

There can be only one central manager in a pool. The central manager is the collector of information and negotiator between resources and resource requests. The central manager is the most crucial node in the pool and needs to be reliable and have good network connectivity with all other nodes in the pool.

8.2 Execute Nodes

Any machine in the pool (including the central manager) can be configured to execute Condor jobs. Execute machines provide the runtime resources for the job (CPU, memory, disk and other runtime dependencies).

8.3 Submit Nodes

Any machine in the pool (including the central manager) can be configured to submit Condor jobs. The resource requirements for a submit machine are much greater than an individual execute node if submitting large number of jobs.

8.4 HTCondor Daemons

The system is geared to provide reliability at high loads while not sacrificing performance. Each of the backing services that run on the nodes are implemented as separate processes, with their own responsibility, fault tolerance and failure semantics as well as clean up mechanism. The list below elaborates on the functions of the more prominent among these (for a complete list of daemons, see the HTCondor Administration Manual):

condor_master: Responsible for keeping all the rest of the HTCondor daemons running on each machine in the pool. Spawns all other daemons on the machine and periodically checks them and respawns them if any has crashed. Also supports administrative commands to start, stop or reconfigure daemons remotely.

condor_startd: This daemon represents a given resource to the HTCondor pool, as a machine capable of running jobs. It advertises certain attributes about machine that are used to match it with pending resource requests. The *condor_startd* will run on any machine in the pool that is to be able to execute jobs.

condor_starter: This daemon is the entity that spawns the HTCondor job on a given machine. It sets up the execution environment and monitors the job once it is running. When a job completes, the condor_starter notices this, sends back any status information to the submitting machine, and exits.

condor_schedd: This daemon represents resource requests to the HTCondor pool. Any machine that is to be a submit machine needs to have a *condor_schedd* running. When users submit jobs, the jobs go to the *condor_schedd*, where they are stored in the job queue. The *condor_schedd* manages the job queue. Various tools to view and manipulate the job queue, such as *condor_submit*, *condor_q*, and *condor_rm*, all must connect to the *condor_schedd* to do their work. If the *condor_schedd* is not running on a given machine, none of these commands will work.

condor_shadow: This daemon runs on the machine where a given request was submitted and acts as the resource manager for the request.

condor_collector: This daemon is responsible for collecting all the information about the status of an HTCondor pool. All other daemons periodically send ClassAd updates to the condor_collector. These ClassAds contain all the information about the state of the daemons, the resources they represent or resource requests in the pool. The condor_status command can be used to query the condor_collector for specific information about various parts of HTCondor. In addition, the HTCondor daemons themselves query the condor_collector for important information, such as what address to use for sending commands to a remote machine.

condor_negotiator: This daemon is responsible for all the match making within the HTCondor system. Periodically, the condor_negotiator begins a negotiation cycle, where it queries the condor_collector for the current state of all the resources in the pool. It contacts each condor_schedd that has waiting resource requests in priority order and tries to match available resources with those requests. The condor_negotiator is responsible for enforcing user priorities in the system, where the more resources a given user has claimed, the less priority they must acquire more resources.

9 HTCONDOR – SETTING UP

9.1 Pre-requisites

The setup instructions below assume the following prerequisites:

a. User has admin access to a network with at least two Linux machines running Ubuntu 18.04 LTS operating system

b. The machines being configured have access to the internet (to download and install required packages)

c. User has access to the Bladed Linux install package

d. User has access to a valid Bladed licence for Linux (this will need to be a networked processor only licence) and that the Linux network being configured to run jobs can access the said licence server

e. User has familiarity with the following tools and technologies

   a. An ssh client

   b. A remote file copy tool (such as scp)

   c. The Linux terminal and basic commands (the OS we are using does not come with a desktop or graphical tools)

   d. A Linux text editor (all commands in this guide use "vi" - one of the built-in text editors, but user may replace it with their preferred option)

   e. Basic understanding of networking concepts (CIDR blocks, firewalls, domain names)

   f. Basic file system concepts (volumes, mounting, file system types)

Note: User may choose to install a desktop (or an X Window or xrdp service) on their worker nodes. However, it is not recommended that they do this because most worker nodes in the pool are typically not directly accessed by users and it would be wasteful to bloat them with such services. If you would like to have more user-friendly way to monitor a pool, a way to achieve this would be to set up a single node which is part of the same pool but setup as a Submit only node but has graphical tools installed. It is also possible to have a Windows machine perform this function.

9.2 Preparation

Create/identify the shared storage space which you will be using to hold Bladed input and output. This could be any networked file store, as long as all nodes in your pool can access it for read and write. Identify a valid Bladed licence for Linux runs and make sure that both test machines can reach it.

9.3 Setting Up A Test Pool

AIM: Set up two machines, one which will act as the "Central Manager" (CM). The other will perform both "Submit" and "Execute" roles

Note: Both nodes will need a series of common setup steps to be followed. Steps below are ordered in the way in which they must be performed. Unless specifically mentioned, perform the steps in the order they appear below on both machines. *All instructions in the guide were written with HTCondor version 8.8.9 and Ubuntu 18.04.4 LTS (Bionic Beaver)*.

Start the two new machines (based on Ubuntu 18.04). Once started, identify one machine which you wish to make the central manager and the other you wish to configure as submit/execute node. Note down the Fully Qualified Domain Names (FQDN) of both nodes.

9.3.1 Prepare the nodes for condor installation

Perform an update and upgrade

```
$ sudo apt-get update && sudo apt-get upgrade
```

Enable password authentication and set password for ubuntu (this is purely to make it easier to work with the nodes). ***This is not a HTCondor specific requirement and need not be performed***. If not performing this, you can skip to the Install HTCondor step.

Edit sshd config and turn on PasswordAuthentication

```
$ sudo vi /etc/ssh/sshd_config
```

Restart the SSH service for the change to take effect

```
$ sudo service ssh restart
```

Set a password for user ubuntu

```
$ sudo passwd ubuntu
```

Note: If you are performing these tests using instances on a Cloud platform, it is possible that you may need to explicitly allow password changes to persist beyond machine lifecycle events such as machine re-imaging. To do this on the AWS platform, editing cloud config file

```
$ sudo vi /etc/cloud/cloud.cfg
```

Change the line "lock_passwd: True" under the default_user section to "lock_passwd: False"

9.3.2 Install HTCondor on both machines

Install the HTCondor repository key

```
$ wget -qO - https://research.cs.wisc.edu/htcondor/ubuntu/HTCondor-Release.gpg.key | sudo apt-key add -
```

Add the HTCondor repositories to the source listing

```
$ echo "deb http://research.cs.wisc.edu/htcondor/ubuntu/8.8/bionic bionic contrib" | sudo tee -a /etc/apt/sources.list
```

```
$ echo "deb-src http://research.cs.wisc.edu/htcondor/ubuntu/8.8/bionic bionic contrib" | sudo tee -a /etc/apt/sources.list
```

Install HTCondor

```
$ sudo apt-get update
$ sudo apt-get install htcondor
```

Note: If you get an option here to run in an assisted mode where HTCondor will attempt to configure the installation based on answers you provide to a series of prompts. **Choose "No"** to proceed with a default install which can then be configured as explained below.

Add a firewall rule to allow HTCodor traffic

```
$ sudo ufw allow 9618/tcp
```

Note: The HTCondor system relies on reliable and timely communication between the nodes that constitutes the cluster. If firewall or other Network ACL rules disallow such traffic, the system would fail to function normally. Signs of such issues include daemons failing to detect members in the pool or being stopped altogether.

Enable HTCondor:

```
$ sudo systemctl enable condor
```

9.3.3 Configure the HTCondor Installation

On both nodes

Set address of the Central Manager node

```
$ echo "CONDOR_HOST = <FQDN for the central manager>" | sudo tee -a /etc/condor/config.d/49-common
```

Setup security settings. Open a file as below and add the content shown into the file.

```
$ sudo vi /etc/condor/config.d/50-security
```

```
SEC_DEFAULT_ENCRYPTION = OPTIONAL
SEC_DEFAULT_INTEGRITY = OPTIONAL
SEC_DEFAULT_AUTHENTICATION = OPTIONAL
SEC_DEFAULT_AUTHENTICATION_METHODS = FS, GSI, KERBEROS, SSL, PASSWORD
SEC_DAEMON_AUTHENTICATION = OPTIONAL
SEC_DAEMON_INTEGRITY = OPTIONAL
SEC_DAEMON_AUTHENTICATION_METHODS = FS, PASSWORD, KERBEROS, GSI
SEC_NEGOTIATOR_AUTHENTICATION = OPTIONAL
SEC_NEGOTIATOR_INTEGRITY = OPTIONAL
SEC_NEGOTIATOR_AUTHENTICATION_METHODS = FS, PASSWORD, KERBEROS, GSI
SEC_CLIENT_AUTHENTICATION_METHODS = FS, PASSWORD, KERBEROS, GSI
ALLOW_READ = *
ALLOW_WRITE = *
```

```
ALLOW_ADMINISTRATOR = *
ALLOW_NEGOTIATOR = *
ALLOW_DAEMON = *
ALLOW_COLLECTOR = *
```

Set filesystem domain and user domain values. Open the local config file as below and add the content shown

```
$ sudo vi /etc/condor/condor_config.local
```

```
FILESYSTEM_DOMAIN = my_shared_filesystem
UID_DOMAIN = <your domain name>
NO_DNS = True
DEFAULT_DOMAIN_NAME = <your domain name>
```

Note: FILESYSTEM_DOMAIN can take an arbitrary string as its value. But, *make sure that you specify the same value on all nodes in the pool* for them to be able to share jobs from a single shared file system. If the values do not match, jobs may be rejected as not matching their ClassAd requirements.

On central manager node only

Set role

```
$ echo "use ROLE: CentralManager" | sudo tee -a /etc/condor/config.d/51-role-cm
```

Set allow-write permissions

```
$ echo "ALLOW_WRITE_COLLECTOR=\$(ALLOW_WRITE) <SUBMIT AND EXEC NODE FQDNs>" | sudo tee -a /etc/condor/config.d/51-role-cm
```

Note: In the above command you can use a * to capture more than one FQDN e.g. ip-10-0-0-*.eu-west-1.compute.internal will allow all machines in the subnet with CIDR 10.0.0.0/24

Disable the STARTD daemon to avoid the central manager running jobs

```
$ sudo vi /etc/condor/condor_config
```

Locate the line that says "DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD, STARTD" and remove the ", STARTD" entry from the end of the line. This will stop the STARTD daemon from spinning up on the central manager, which will avoid jobs executing on the central manager. If this entry is not found, create an entry "DAEMON_LIST = COLLECTOR, MASTER, NEGOTIATOR, SCHEDD" as a new line at the end of the file.

Note: It is possible to keep running jobs on the central manager, but it will then require the user to make sure that versions of Bladed Linux that are used are also installed on the central manager, as well as the shared file system is mounted on to the central manager. This guide assumes that the central manager will not perform job execution.

On submit/exec node only

```
$ echo "use ROLE: Submit,Execute" | sudo tee -a /etc/condor/config.d/51-role-submit-exec
```

9.3.4 Start HTCondor

On All nodes

```
$ sudo systemctl start condor
```

Check that it is running

```
$ sudo systemctl status condor
```

Test basic HTCondor commands

See the status of the queue

```
$ condor_q
```

Sample output from the command:

```
ubuntu@ip-10-0-0-241:~$ condor_q


-- Schedd: ip-10-0-0-241.eu-west-1.compute.internal : <10.0.0.241:9618?... @ 06/11/20 13:00:58
OWNER BATCH_NAME      SUBMITTED   DONE   RUN    IDLE   HOLD  TOTAL JOB_IDS

0 jobs; 0 completed, 0 removed, 0 idle, 0 running, 0 held, 0 suspended
```

Check status of slots and nodes

```
$ condor_status
```

Sample output from the command:

```
ubuntu@ip-10-0-0-241:~$ condor_status
Name                                OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

ip-10-0-0-241.eu-west-1.compute.internal LINUX      X86_64 Unclaimed Idle      0.030   978  0+00:49:47

                     Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain

       X86_64/LINUX     1     0       0         1       0          0        0      0

              Total     1     0       0         1       0          0        0      0
```

9.4 Installing and Configuring Bladed

Preparation:

Copy the Bladed Linux installer package (e.g. dtbladedlinux-4.10.0.21.deb) on to the submit/exec node

Install dtbladed

```
$ sudo dpkg -i dtbladedlinux-4.10.0.21.deb
```

Note: The Bladed install will prompt users to perform installation of the Sentinel runtime environment. Follow instructions provided by the Bladed install to perform this step which completes the Bladed installation on the machine.

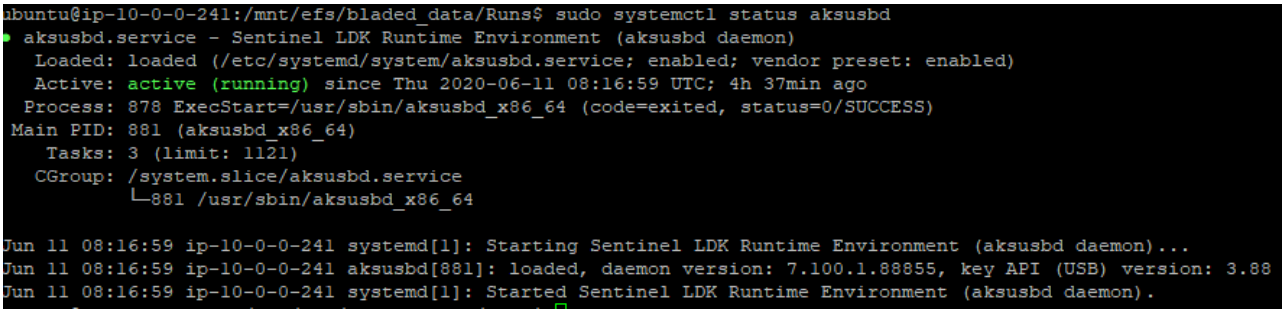
Start the Sentinel runtime service

```
$ sudo service aksusbd start
```

Verify the runtime installation

```
$ sudo service aksusbd status
```

Sample output:



Verify dtbladed installation

```
$ dtbladed-4.10.0.21
```

Expected output at this stage:

Point the licence local runtime to point to a valid licence server. To do this, create a config file for the runtime and populate it with below content. **Note**: You will need to update the serveraddr attribute's value in the content below based on the actual licence server you are going to use within your environment.

```
$ sudo vi /etc/hasplm/hasplm.ini
```

```
;*********************************************************************
;*
;* Sentinel License Manager configuration file
;*
;* Version 20.0 1.70826 at bladed-linux-gw
;* Tue, 20 Aug 2019 10:31:28 GMT
;*
;*********************************************************************
[SERVER]
name = bladed-linux
pagerefresh = 3
linesperpage = 20
ACCremote = 0
enablehaspc2v = 0
old_files_delete_days = 90


enabledetach = 0
```

```
reservedseats = 0
reservedpercent = 0
detachmaxdays = 14
commuter_delete_days = 7
disable_um = 0

requestlog = 0
loglocal = 0
logremote = 0
logadmin = 0
errorlog = 0
rotatelogs = 0
access_log_maxsize = 0 ;kB
error_log_maxsize = 0 ;kB
zip_logs_days = 0
delete_logs_days = 0
pidfile = 0
passacc = 0

accessfromremote = 0
accesstoremote = 1
bind_local_only = 0  ; 0=all adapters, 1=localhost only

[UPDATE]
download_url = www.safenet-inc.com/hasp/language_packs/end-user
update_host = www3.safenet-inc.com
language_url = /hasp/language_packs/end-user/

[REMOTE]
broadcastsearch = 0
aggressive = 0
serversearchinterval = 30
```

```
serveraddr = <Enter server IP address/hostname here>


[ACCESS]


[USERS]


[VENDORS]

[EMS]

emsurl = http://localhost:8080

emsurl = http://127.0.0.1:8080

[LOGPARAMETERS]

text = {timestamp} {clientaddr}:{clientport} {clientid} {method} {url} {function}({functionparams})
result({statuscode}){newline}
```

Tip: If you get multiple tab characters when copying in content from this page, you will need to replace them. If you are using the "vi" editor as shown in the example, you can do that by running the vi replace command **%s/\t\t//g**. You can get the "vi" command prompt by pressing the ":" key when in command mode (not in insert mode).

Note: Once you have added this config, it can take up to a couple of minutes for the local Sentinel Runtime to detect the key from the server.

Verify that the licence server was found by running dtbladed again

```
$ dtbladed-4.10.0.21
```

Expected output at this stage:

```
ubuntu@ip-10-0-0-241:~$ dtbladed-4.10.0.21
ERROR:  Bladed input data: The file DTBLADED.IN does not exist.
Run completed normally.
```

9.5 Creating a shared file system

Note: This is a step that is very much dependent on the local setup within which the user is operating. It is recommended that the user gets help from local IT or sysadmin to identify the best approach for having a shared file system across the nodes in the HTCondor pool if they don't already have a mechanism in place.

For purposes of illustration, steps below show how an EFS volume (Amazon Elastic Filesystem) could be mounted onto a Submit/Exec node. These steps are unlikely to be applicable to most user scenarios, however, the basic workflow wold be the same. Also note that

Create an EFS volume in AWS (see https://docs.aws.amazon.com/efs/latest/ug/gs-step-two-create-efs-resources.html for detailed instructions). Note that EFS volumes are associated with a VPC, so make sure to associate it with the VPC which hosts the subnet where the HTCondor pool will reside. You will need the fully qualified name of this volume later to mount it.

Create a mount point on your HTCondor Submit/Execute node

```
$ sudo mkdir /mnt/efs/
```

Install packages to provide support of EFS volume mounts using NFS4

```
$ sudo apt-get install nfs-common
```

Edit the fstab to add mount instructions

```
$ echo "fs-90916d5a.efs.eu-west-1.amazonaws.com:/ /mnt/efs nfs4
nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2 0 0" | sudo tee -a /etc/fstab
```

Mount the shared file system

```
$ sudo mount -a
```

Create a folder hold test bladed runs within the shared folder

```
$ cd /mnt/efs && sudo mkdir bladed_data && sudo chown ubuntu:ubuntu bladed_data
```

```
$ mkdir bladed_data/Runs
```

9.6 Creating Sample Runs

Now that we have a simple HTCondor pool, working dtbladed installation and a shared file system to host the runs, the next task is to create some test runs and to validate the setup.

Note: Steps below assume a shared folder at /mnt/efs/bladed_data/Runs. If your file system layout is different, please amend the paths used in following commands before you run them.

A runs creator utility can be downloaded from the software portal here. This utility will help you quickly create some test runs to verify your setup. Note that you will need to login to the portal for the download link to function.

Download and copy the supplied "sample_runs_creator.tar.gz" file to the /mnt/efs/bladed_data/Runs folder on the submit/exec node. Once copy is complete, unpack the archive to get files that we will use to create some sample runs.

```
$ cd /mnt/efs/bladed_data/Runs && tar -xzvf sample_runs_creator.tar.gz
```

This will unpack the archive into the current folder. See table below for folders/files in the package and their role.

	Type	**Description**
base-run/	Folder	Folder containing a template DTBLADED.IN and submit description files.
base-run/DTBLADED.IN	File	This file is based on the Bladed demo project, but with the difference that it uses a constant wind of 12 m/s. The RUNNAME and PATH variables in the ".IN" file contain tokens which can replaced with appropriate values by a script to create a set of new runs which can be used to experiment

		with your HTCondor pool.
base-run/runs.sub	Submit description file	A HTCondor submit request file that specifies how to run the sample jobs. The file encapsulates details including the executable to invoke, the way to run it, the environment around it etc. See https://htcondor.readthedocs.io/en/latest/man-pages/condor_submit.html?highlight=submit%20file#submit-description-file-commands for a complete definition of what can be defined through this file. This file is used as a template just the same way the DTBLADED.IN **Note**: The supplied runs.sub file uses Bladed 4.10.0.21 by default. If you would like to test with a different version of Bladed, please update the "**executable**" parameter value in this file to make your jobs use the version you would like to test with.
utils/	Folder	Folder with utility scripts to generate actual runs and submit descriptions based on template files in base-run folder
utils/createABladedRun.sh	Executable shell script	A script to create a single named run based on the template DTBLADED.IN file in base-run. Name of the run to be passed in as an argument.
utils/createSubmitDescription.sh	Executable shell script	Create a submit description file based on the template "runs.sub" file in base-run folder. Takes two arguments, a base run name and total number of jobs to queue.
create_runs.py	Python script	A simple python program that wraps the utility scripts to create a fixed number of runs and a submit description for it. To be invoked with two arguments: first, the total number of jobs to create and second, a base name to use for the individual runs. E.g. the following command creates 2 sample runs (foo-0 and foo-1) in the current directory based on the base-run definition. `$ python create_runs.py 2 foo` In addition, the script will also produce a submit description file named "foo.sub" which can be used to submit the runs to the condor pool.

Create two sample runs to test HTCondor with

```
$ python create_runs.py 2 foo
```

Check that this command has produced two folders in the current working directory named "foo-0" and "foo-1" as well as a submit file named "foo.sub". Within each run folder you should fine a DTBLADED.IN file as well as a "results" folder

9.7 Submitting jobs to the pool

Now that we have the sample runs, we can request HTCondor to run them. To submit the jobs, run the following command

```
$ condor_submit foo.sub
```

Check the status of the jobs you submitted

```
$ condor_q
```

Expected output for a single exec node with 1 CPU

```
ubuntu@ip-10-0-0-241:/mnt/efs/bladed_data/Runs$ condor_q


-- Schedd: ip-10-0-0-241.eu-west-1.compute.internal : <10.0.0.241:9618?... @ 06/11/20 13:14:36
OWNER  BATCH_NAME          SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
ubuntu bladed_batch:foo   6/11 13:14      _      1       1      2 31.0-1

2 jobs; 0 completed, 0 removed, 1 idle, 1 running, 0 held, 0 suspended
```

Note that one job has transitioned to the "Running" stage while 1 is "Idling". This is because, the node used in the test setup was a single CPU machine, which in HTCondor terms only provide a single job processing slot, which translates to a default concurrency of 1. If you have used a machine with > 1 core, then you may find at this stage that both jobs transition to the "Running" stage simultaneously. Wait for about 40 seconds (for first job to finish) and run the "***condor_q***" command again. This time you should see that 1 job has moved to the "Done" column and the previously idling job is now "Running".

Verify the results from the run. Within each run (e.g. "run-0") you should see the following files.

	Type	**Description**
<runname>.$PR	Bladed progress file	Produced by dtbladed. Shows the simulation's progress as a percentage
<runname>.$ME	Bladed message file	Produced by dtbladed. Contains all messages produced by the simulation
<runname>.$TE	Bladed termination file	Produced by dtbladed. Contains the termination status (Success/Error) E.g. "Run completed successfully", "Run completed

		(with 2 warning"), "Run terminated ERROR: <error_message>"
out.txt	Output	Produced by HTCondor. Contains all messages written to the standard output stream by the simulation. This will also include any messages printed out to stdout stream by user components (such as external controllers or other dynamically loaded modules)
error.txt	Output	Produced by HTCondor. Contains all messages written to the standard error stream by the simulation or user components.
<runname>.log	HTCondor log for the run	Produced by HTCondor. Contents include Submission details (time, source of submission) Execution details (time, execution location) Program termination details (CPU time, return code) Resource usage stats (CPU, Memory, Disk)
results	Folder	This folder would contain the results from the run. Note that the location of the results folder is entirely dependent on the PATH variable specified by the input file (DTBLADED.IN). The sample jobs created as part of this guide use a "results" folder within each run. This folder should contain bladed output files for the run (which appear as % and $ file pairs, with each % file representing a unique output group and the corresponding $ file representing the data for that group).

Check the contents of some of the files to make sure they represent a successful run. For example, the $PR file should contain the value "100" and the out.txt file should show data including the dtbladed "banner" that is typically printed out to the console.

9.8 Scaling Up the Pool

Now that you have a functioning HTCondor pool, it is time to replicate the submit/exec node setup to multiple computers (real or virtual) to increase the size of the pool. The exact steps for performing this task depends on the platform being used. Note that since only submit/exec nodes perform the actual calculations, it is only necessary to replicate them (and not the central manager).

For a virtualized setup (such as for example, a set up using a Cloud platform), you could save the current Submit/Exec node's state as a new machine image and start up the required number of worker nodes based on the new image. All new nodes should automatically connect to the existing central manager and be ready to pick up jobs or to submit jobs. The fstab entry made in the setup process will also ensure that the necessary file system access is available on nodes replicated in this way.

Once you have increased the HTCondor pool size in this manner, check the status of the pool by running the ***condor_status*** command from an ssh session connected to any of the machines in the pool (either from any of the Submit/Exec nodes or from the Central Manager). Output from the command should give you an up to date view of the pool and the available processing slots including their current status. Image below shows the output from the ***condor_status*** command for a pool with 11 processing slots spread across 7 machines (4 nodes with 2 CPUs and 3 nodes with a single CPU each). Nodes with multiple CPUs are listed as slot<n>@hostname whereas single processor machines are identified simply by their hostname.

```
Name                                     OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

ip-10-0-0-22.eu-west-1.compute.internal  LINUX      X86_64 Unclaimed Idle      0.000  978  0+00:38:35
slot1@ip-10-0-0-45.eu-west-1.compute.internal  LINUX  X86_64 Unclaimed Idle    0.000 1966  0+04:07:46
slot2@ip-10-0-0-45.eu-west-1.compute.internal  LINUX  X86_64 Unclaimed Idle    0.000 1966  0+04:07:46
ip-10-0-0-116.eu-west-1.compute.internal LINUX      X86_64 Unclaimed Idle      0.000  978  0+00:55:42
slot1@ip-10-0-0-173.eu-west-1.compute.internal LINUX  X86_64 Unclaimed Idle    0.000 1966  0+04:08:03
slot2@ip-10-0-0-173.eu-west-1.compute.internal LINUX  X86_64 Unclaimed Idle    0.000 1966  0+04:07:34
ip-10-0-0-241.eu-west-1.compute.internal LINUX      X86_64 Unclaimed Idle      0.000  978  0+00:34:47
slot1@ip-10-0-0-248.eu-west-1.compute.internal LINUX  X86_64 Unclaimed Idle    0.000 1966  0+04:08:11
slot2@ip-10-0-0-248.eu-west-1.compute.internal LINUX  X86_64 Unclaimed Idle    0.070 1966  0+04:08:12
slot1@ip-10-0-0-250.eu-west-1.compute.internal LINUX  X86_64 Unclaimed Idle    0.000 1966  0+04:07:47
slot2@ip-10-0-0-250.eu-west-1.compute.internal LINUX  X86_64 Unclaimed Idle    0.000 1966  0+04:07:48

                     Total Owner Claimed Unclaimed Matched Preempting Backfill  Drain

        X86_64/LINUX    11     0       0        11       0          0        0      0

               Total    11     0       0        11       0          0        0      0
```

Having increased the pool size, use the sample runs creation scripts to produce a larger batch of jobs (say 100 jobs) and submit it to the pool. Observe the status of the pool and the queue using the commands ***condor_q*** and ***condor_status***.

For example, to create 100 jobs using the script, go to the location where you unpacked the sample runs creator in the step "Create sample runs".

```
$ python create_runs.py 100 scaled-batch-1
```

```
$ condor_submit scaled-batch-1.sub
```

```
$ condor_q
```

```
$ condor_status
```

You can repeat the last two commands to see the jobs moving through their states from "Idle" to "Running" to "Done".

You may also create a second batch at the same time as the first one is processing and submit it.

```
$ python create_runs.py 100 scaled-batch-2
```

```
$ condor_submit scaled-batch-2.sub
```

```
$ condor_q
```

Note that you should now be able to see status for both batches as separate entries in the output

10 NEXT STEPS

As indicated in the Scope and Limitations sections of this guide, certain topics relating to the setup and configuration of an HTCondor cluster are not dealt with in detail in this guide. For instance, to provide an easier start to the process, the guide uses an all permissive security model and does not address the issue of multi-user access management. The setup you have performed by following this guide is best treated as a proof of concept, allowing you to understand the moving parts of the system and to gain an understanding of how your existing workflow may need reshaping in order to fit with the batch processing mechanism. Once it becomes apparent how you would like your production environment to be, you should consider performing a fresh setup with additional configuration as needed.

HTCondor is a complex product which satisfies many usage scenarios. It is therefore highly recommended that users of this guide who have made the decision to incorporate it into their production workflow spend more time familiarising themselves with the product.

See resources below for further help on setting up and configuring HTCondor:

Installation (for various platforms):
https://agenda.hep.wisc.edu/event/1325/session/16/contribution/41/material/slides/0.pdf

Latest User Manual:
https://htcondor.readthedocs.io/en/latest/users-manual/index.html

Admin Basics (and rules of thumb):
https://indico.cern.ch/event/467075/contributions/1143813/attachments/1236271/1815391/Admin_Basics_Condor_Week_Barcelona_2016.pdf

Admin Tutorial:
https://research.cs.wisc.edu/htcondor/CondorWeek2009/condor_presentations/admin_tutorial/