Using the NA62 DAQ farm for offline processing by exploiting its computational power

CERN SUMMER STUDENT 2019 PROGRAM

ABDULLATIF RASHDAN | SUPERVISED BY: MARCO BORETTO | 16TH Aug 2019
## Table of Contents:

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectives</td>
<td>2</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>2</td>
</tr>
<tr>
<td>2. High Throughput Computing</td>
<td>2</td>
</tr>
<tr>
<td>2.1 High Throughput Computing (HTC) vs High Performance Computing (HPC)</td>
<td>2</td>
</tr>
<tr>
<td>2.2. HTCondor</td>
<td>3</td>
</tr>
<tr>
<td>2.2.1 Machine Roles</td>
<td>3</td>
</tr>
<tr>
<td>2.2.2 User Guide</td>
<td>5</td>
</tr>
<tr>
<td>3. Exploiting the Computational Power using HTCondor</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Main Components</td>
<td>7</td>
</tr>
<tr>
<td>3.2 CC7-Condor Image</td>
<td>8</td>
</tr>
<tr>
<td>3.3 Installing HTCondor as execute node inside the container</td>
<td>11</td>
</tr>
<tr>
<td>3.4 Mounting CVMFS inside the container</td>
<td>11</td>
</tr>
<tr>
<td>3.4 Installing EOS</td>
<td>11</td>
</tr>
<tr>
<td>4. Conclusion</td>
<td>13</td>
</tr>
<tr>
<td>4. Bibliography</td>
<td>14</td>
</tr>
<tr>
<td>Appendix A – Example on execute node configuration file</td>
<td>15</td>
</tr>
</tbody>
</table>
Objectives:

- Design a generic solution that can be deployed on any cluster
- Design a solution that is logically isolated from the environment
- Create the proper documentation to make the work reproducible

1. INTRODUCTION:

During the long shutdown 2 (Known as LS2) in CERN that take place from 2018 to 2020, many experiments including the NA62 experiment enter the shutdown status where experiments undergone development, maintenance and upgrades. During that time, many computational resources are not being exploited and just being idle.

The NA62 experiment is a fixed target experiment that investigate the rare Kaon decay. It is good to know that this experiment is also known by the name Kaon factory. The collaboration has its own PC Farm for the DAQ system (Data acquisition system). It consists of thirty servers and three mergers. The aim of the thirty servers is to process the raw data generated from the collisions and process them in software level. After that, the data is being received by the mergers which send them to the permanent storage in CERN.

Since this experiment and many others are included the LS2. Although there are some tests being conducted from time to time but most of the time their computational power remains unexploited. In this project we aimed to exploit the computational power of the PC farm or any unused infrastructure by deploying and configuring HTCondor in a way that it will be isolated from the environment and generic. Hence, minimal changes to these infrastructures will be done and it can be rapidly deployed anywhere.

2. HIGHT THROUGHPUT COMPUTING:

2.1 High Throughput Computing (HTC) vs High Performance Computing (HPC)

HTC is the computing field concerned with number of computational jobs that can be done over a long period of time. Hence the name “High throughput”. In contrary to the high-performance computing (HPC) which is concerned with how fast can operation be done or how many cycles per second. HTC can be seen as a quantitative approach to computational tasks over a long period of time whereas HPC focuses on large computational power over short period of time.

Since the beginning, the old approach was to have a centralized main frame in the facility (known as supercomputer) that will execute operations as fast as possible and will consume significant amount of power. When smaller workstations evolved to have a very decent computational power and become accessible to everyone. HTC become a very good idea, since these machines are not always up and running CPU intensive tasks. Therefore,
grouping all these smaller workstations along with other unexploited machines into one pool that can be used to execute big amount of jobs over long period of time. Therefore, this is one of the most important characteristics of HTC. For example, you can add your classroom personal computers and dedicated servers into one pool using HTC. HTC requires management, scheduling, queuing, error handling and many other features. If HTC was done manually it would be a very tough job to go through every machine and execute your job in it. For that reasons people developed HTC systems that will make the operations of submitting, executing and transferring results back automated.

2.2 HTCondor

HTCondor is an open-source HTC system that was developed by University of Wisconsin-Madison. It provides all the necessary features for the user to submit jobs and administrators to manage their pools of computational resources.

“HTCondor is a specialized workload management system for compute-intensive jobs. Like other full-featured batch systems, HTCondor provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. Users submit their serial or parallel jobs to HTCondor, HTCondor places them into a queue, chooses when and where to run the jobs based upon a policy, carefully monitors their progress, and ultimately informs the user upon completion.” [1]

From the user side, users can submit jobs including all necessary files, keep track and receive log files reporting the status of the jobs. From an admin side, admins can easily setup their own pool and manage different machines. They can prioritize users, prioritize machine, chose the different roles for machines and many other features that HTCondor allows you to do.

2.2.1 HTCondor: Machine Roles

In this section we will discuss how admins design their pool and what are the different possible machine roles that can be in any HTCondor pool. In HTCondor there are three main machine roles that a machine can be. The roles are central manager (CM), submit machine and execute machine. Keep in mind that these roles can be all integrated into one node or split in different ways.

The central manager (CM) has two main tasks which are collecting Machine class add and matchmaking between machines and jobs. There can only be one CM in any HTCondor for each pool. Machine class add is a brief description about the machine capabilities and it is transmitted over the networks by every execute node periodically. It contains useful information that will be used in the process of matching such as version of compilers, interpreters or whether the machine has docker engine installed or not. The central manager will collect these ads and perform matching between the ads and the jobs being submitted by users. The process of collecting adds and matchmaking is done by two
demons condor_collector and condor_negotiator respectively. The CM must be a reliable machine because once this machine crashes there is no further matchmaking which can disturb the normal workflow in the pool.

Submit machine can be any machine in the pool. These machines require much greater resources than any other role. For two main reasons, some type of jobs will be transferred to the local disk of this machine which implies requirements in storage space in the submit machine. The other reason is that the submit machine keep track of every remote job by running a process for every job that is being executed remotely on execute node.

Execute machines as the name suggests are used for executing jobs in the pool. There must be at least one machine, or the pool will not serve the purpose of “executing jobs”.

In Figure 1, you will find the configuration of HTCondor on a single node also known as Minicondor. Minicondor is a type of installation that includes all the roles inside one machine. All the discussed roles and related demons will communicate with each other using the loopback interface. In Figure 2 is another type of configuration where the execute node is being separated from submit and CM. In Figure 3, the configuration where each machine has its own role. As you can see there is always only one central manager, and fewer submit machines than execute machines. Figure 2 and 3 are mostly used in real life scenario where big pool is to be setup.

Figure 1: One example of HTCondor configuration known as Minicondor
2.2.2 HTCondor: User Introduction

From user side, users are able to submit jobs by installing HTCondor client on their personal machines and create a submit directory that will consist of submit description file, input files and executable. In the submit description file user can specify important parameters such as the executable location, whether to transfer files or not, which machine in the pool is preferred to be used, arguments supplied, log and error files, or input files.
In the box below is one of the possible submit description file that will give you an idea of how submit file works.

```plaintext
executable = ./exe.sh
should_transfer_files = YES
rank = ((machine == "slot1@na62condor-2.cern.ch") || (machine == "slot2@na62condor-2.cern.ch") || (machine == "slot3@na62condor-2.cern.ch") || (machine == "slot4@na62condor-2.cern.ch") || (machine == "na62condor-2.cern.ch"))
Universe = Vanilla
Log = myexe.log
Error = error_job.log
input = user.key
when_to_transfer_output = ON_EXIT
output = reco_c10_dr6610_r396043.root
```

In the following submit file there are some nice features that can become useful in many situations such as:

1- PeriodicRemove = ( JobStatus == 2 ) && ( ( CurrentTime - EnteredCurrentStatus ) > 90

This line will limit the job duration to 90 seconds. If the job is not done by that time, it will be terminated.

2- Queue 100

This line will queue 100 job of the same type for execution.

3- rank = ((machine == "slot1@example.cern.ch") || (machine == "slot2@example.cern.ch") || (machine == "slot3@example.cern.ch") || (machine == "slot4@example.cern.ch") || (machine == "example.cern.ch"))

By using the rank command users can requests many parameters such as memory or specific machine. In the above line you can see us targeting a specific machine in the pool.

4- should_transfer_files = YES

This line will force HTCondor to transfer all files from submit directory to the execute directory on the remote machine. This makes the job independent from the user. This can become in handy if the machine who submitted the job become unavailable temporarily or network failure occur. Since we are making the job execution totally independent from the machine who submitted it.
Users can navigate to the submit directory where condor client is installed and enter the following command:

```
condor_submit SUBMIT_FILE_NAME
```

In this section we discussed briefly the submit description file. We omitted other parameters in the file because the aim of this section is to briefly introduce how users are supposed to submit jobs. For more details consult the HTCondor documentation.

### 3 Exploiting the Computational Power using HTCondor

As discussed in the introduction, the aim of this project is to exploit the computational power of the unused infrastructure at CERN and mainly the NA62 PC farm during the LS2. The exploitation of this infrastructure is done by deploying a containerized HTCondor execute node that has all the necessary computing tools such EOS and CVMFS to recreate CERN computing environments. By this we can convert any unused infrastructure into a functioning HTCondor execute node that is ready to receive, connect to different services in CERN and execute.

HTCondor by itself is not totally useful in CERN environment where scientists use many sub-tools and storage services to perform computational tasks. The challenge was to recreate CERN environment in an isolated container that will transform any machine to an operational execute node with all the features needed by CERN staff to execute jobs and once this node is needed to be used by CERN we just remove the container.

The solution was containerized inside a docker container to make it portable and to ease the deployment and removal of the execute nodes because most of these unused infrastructures will become fully operational after the LS2 ends. Another advantage of containerizing the solution is to avoid missing or modifying the environment of which the execute node is deployed.

#### 3.1 Main components

In this project, we aimed to recreate CERN computing environment by integrating all the components that are necessary to make HTCondor useful. This is the list of all the components where used and their tasks in the project:

- CentOS7: light Linux image of CC7 which designed for docker containers. On top of this image we installed the rest of the tools.
- CernVM File System: This service provides reliable, scalable software distribution service. It is used to distribute software over the grid. In this project, it was mounted locally on the host machine, then mounted inside the docker container to be able to access the grid software used in CERN
- EOS: Storage service used to access the data for offline processing.
• HTCondor: High throughput computing tool used with EOS and CVMFS to submit and manage jobs.
• Docker: To create, run and terminate the containers.

### 3.2 CC7-Condor image

![Diagram of CC7-Condor image]

CC7-Condor is the main result of this project. It is a containerized execute node that can be easily created and removed. The only application needed to run is the docker engine. This docker image comes with all the dependencies required to recreate CERN computing environment.

In figure 3 is a rough diagram to summarize what we underdone during the development of CC7-condor image. The host OS which can be any host that has docker engine installed on it. On the next layer is the Cent OS 7 base image and then we configured HTCondor to work as execute machine and communicate with EOS and CVMFS successfully. This execute node once created will advertise its existence to the CM and it will be added to the pool. The insertion process is not immediate, it requires time since the updates are being sent periodically. EOS here is used to access different experiments or user data which is required in almost all computational tasks. CVMFS is used to access the grid software distribution to have one standard software versions. Therefore, HTCondor is reading from EOS and executing using the distributed grid software.
The advantage of this is that it is easy to create and manage one CM and submit machine. But when it comes to thousands of execute nodes and configuring them it can become tedious. Therefore, we wrote a Docker file and bash script that will build and run the container automatically with all its dependencies and check if CVMFS is mounted locally otherwise, it will mount it.

Example of the build.sh file:

```bash
#!/bin/bash
if [ ! -d "/cvmfs" ]; then
    # Control will enter here if $DIRECTORY doesn't exist.
    echo 'CVMFS directory not found. Mounting ..'
    sudo yum install -y cvmfs cvmfs-config-default
    cd /etc/cvmfs/
    echo 'CVMFS_HTTP_PROXY="http://ca-proxy.cern.ch:3128"' > default.local
    sudo mkdir -p /cvmfs/sft.cern.ch
    sudo mount -t cvmfs sft.cern.ch /cvmfs/sft.cern.ch
    echo 'CVMFS is mounted'
    cd -
else
    echo 'cvmfs is already mounted. Proceed with creating the execute node'
fi
#building container
export DOCKER_OPTS="--dns DNS_SERVER_IP_ADDRESS --dns DNS_SERVER_IP_ADDRESS"
service docker restart
echo 'Building container...'
docker build -t exnode .
echo 'Type ./run.sh to start the container...'
```
Example of the Dockerfile:

FROM gitlab-registry.cern.ch/linuxsupport/cc7-base

RUN yum install -y htop
RUN yum install -y wget
RUN wget https://research.cs.wisc.edu/htcondor/yum/RPM-GPG-KEY-HTCondor
RUN rpm --import RPM-GPG-KEY-HTCondor
RUN yum install -y condor
EXPOSE 9618
COPY 00-config /etc/condor/config.d/

#EOS Installation
COPY user.key /home/
RUN kinit -kt /home/user.key arashdan
RUN yum install -y locmap
RUN yum install -y locmap && locmap --enable eosclient; locmap --configure eosclient || true
RUN export EOS_MGM_URL=root://eosna62.cern.ch
RUN ( echo text ; echo "export EOS_MGM_URL=root://eosna62.cern.ch" ) >> ~/.bashrc
WORKDIR /root
ENTRYPOINT condor_master && tail -f /dev/null && source ~/.bashrc

Many technical issues were resolved in this design mainly communication and authentication issues. In order to have successful communication between a containerized execute node and Central manager one should allow the following policies:

1- import the host interface as the docker container interface otherwise expose port 9618 which is used by HTCondor for communication
   o Note: In case you did not import the host interface inside the container make sure you perform port mapping between host 9618 and container 9618
2- Copy execute node configuration file to the container and specify your central manager IP or host name.
3- Finally, do not forget to allow port 9618 on the host firewall.

Most of the tools were not designed to work inside the container. Many solutions were made to make their working inside the container possible. In the following section, we will discuss how HTCondor, EOS and CVMFS were installed inside the container.
3.3 Installing HTCondor as execute node inside the container

Installing HTCondor inside the container is almost a straightforward process. It is included in the first few lines of the Dockerfile. The important part is to copy the machine specific configuration file of the execute node to the container HTCondor path "/etc/condor/config.d". An example of this configuration file is included in appendix A.

```
RUN yum install -y wget
RUN wget https://research.cs.wisc.edu/htcondor/yum/RPM-GPG-KEY-HTCondor
RUN rpm --import RPM-GPG-KEY-HTCondor
RUN yum install -y condor
#This line can be ignored if you import the host interface inside the container
EXPOSE 9618
#Copy machine-specific configuration
COPY 00-config /etc/condor/config.d/
```

The last line will copy the machine-specific configuration inside the container to be used by HTCondor to work as execute node and communicate with CM.

In order to keep the container running and run condor_master which is the main demon in HTCondor we used the following line:

```
ENTRYPOINT condor_master && tail -f /dev/null && source ~/.bashrc
```

3.4 Mounting CVMFS inside the container

As shown in the build.sh above. This script will check if CVMFS directory does not exist it will mount it locally on the host. The last step is to add the host volume inside the container when running the container.

```
#!/bin/bash
docker run -d -v /cvmfs/:/cvmfs/:shared --network=host -h execute_node --name execute_node -p 9618:9618 execute_node
```

3.5 Installing EOS

EOS is a storage service by CERN to access data. The installation of this service on CentOS7 is direct using the following commands:

```
RUN yum install -y locmap && locmap --enable eosclient; locmap --configure eosclient || true
RUN ( echo text ; echo "export EOS_MGM_URL=root://eosna62.cern.ch" ) >> ~/.bashrc.
```
The first line will configure EOS client while the second will export EOS_MGM_URL environment variable inside the container inside the bashrc file which will make it persist inside the container.

In order to allow access to these files stored in EOS, users must be authenticated using Kerberos tickets inside CERN network or X509 certificate if on the grid and based on their level of authorization they will be granted access to certain files. This gave the rise of new challenge which is authenticating CERN username inside the CC7-Condor container.

Once we are inside the container, we no longer function as CERN users unlike LXPLUS or other services provided and managed by CERN centrally. The solution to overcome this problem is to authenticate CERN users using a key tab file. Key tab files are used are used to authenticate on behalf of the user. Any service that has access to your keytab file will be able to generate a valid Kerberos ticket by your login name. That is why it is important to keep this file in private place. In our case, within the executable that will be executed by HTCondor we include the following line:

```
 kinit -kt ./user.key LOGIN
```

By this, we acquired a valid credentials and we can access EOS. Keep in mind, that any service posses this file will be able to authenticate on your behalf. Therefore, we advise you keep this file private. In order to generate this key tab file CERN has a service called cern-get-keytab [2] that will generate a key tab file based on your login name after that it will prompt you to enter your password.

The procedures to create key tab file is as follow:

```
 ktutil
 wkt user.key
 exit
 cern-get-keytab --user --login
 LOGIN --keytab ./user.key
```

- Replace LOGIN with your CERN user login

We copied the user.key during the creation of the docker container by executing the following line the docker file:

```
 COPY user.key /home/
 RUN kinit -kt /home/user.key LOGIN
```
4. Conclusion

In this project, we were able to successfully design a model for a containerized HTCondor execution node integrated with CernVM FileSystem and EOS Storage service to recreate the CERN computing environment. It is necessary for this execute node to be portable, easily deployed and removed for making it more convenient to use during the long shutdown 2 and removed afterwards. This case can be more generalized for the all unused infrastructure in CERN which can now be easily exploited and transformed into a HTCondor execution node. We focused on making the process of building and mounting other services into the container all done via Linux scripts that will make this solution generic and reproducible. In fact, there can be possibility in the future to automate the deployment of these portable containers through cloud orchestration. Instead of running the bash script manually users can deploy thousands of HTCondor nodes using Kubernetes.
5. BIBLIOGRAPHY


Appendix A. Example on execute node configuration file

```
#What machine is your central manager?
CONDOR_HOST = example.cern.ch

## This macro determines what daemons the condor_master
## will start and keep its watchful eyes on.
## The list is a comma or space separated list of
## subsystem names
#Daemon that will run on the machine
DAEMON_LIST = MASTER, STARTD
#Role of your machine
use ROLE = Execute

# Verify authenticity of HTCondor services by checking if
# they are running with an effective user id of user "condor".
ALLOW_DAEMON = condor@$(UID_DOMAIN) example.cern.ch
ALLOW_NEGOTIATOR = condor@$(UID_DOMAIN) example.cern.ch

# Configure so only user root or user condor can run
# condor_on, condor_off, condor_restart, and condor_userprio commands
to manage
# HTCondor on this machine.
# If you wish any user to do so, comment out the line
# below.
ALLOW_ADMINISTRATOR = localhost
```