HTCondor-CE at CC-IN2P3

C. Eloto, N. Fournials, E. Vamvakopoulos and V. Hamar
Overview

- CC-IN2P3
- HTCondor-CE
  - Deployment
  - Configuration
  - External grid nodes configuration
  - Monitoring
- HTCondor
  - Configuration at CC
- Future Work
- Conclusions
Centre de Calcul de l’IN2P3 / CNRS
Established in Villeurbanne since 1986

Missions
- Mass storage and computing infrastructure
- Network and connectivity
- Common and collaborative services
Centre de Calcul de l’Institut National de Physique Nucléaire et de Physique des Particules
Architecture

Our choice

HTCondor_CE cernops module

- Customized to fit our needs
  - BDII deployment added
  - A few parameters changed
  - Deploy a CE is easy and reproducibly.

https://github.com/cernops/puppet-htcondor_ce

Used in conjunction with HTCondor HEP-Puppet module

- Very useful deployment tool for HTCondor

https://github.com/HEP-Puppet/htcondor
htcondor_ce::pool_collectors:
   - 'cccondorcm01.in2p3.fr'

htcondor_ce::condor_view_hosts: []
htcondor_ce::ce_version: '3.3.0-1.el7'
htcondor_ce::lrms_version: '8.6.12-0.445603.el7'
htcondor_ce::uid_domain: 'in2p3.fr'
htcondor_ce::gsi_regex: '^\O=GRID-\FR\C=FR\C=CNRS\OU=CC-IN2P3\CN=([A-Za-z0-9.\-]*)$'
htcondor_ce::gsi_backend: 'argus'
htcondor_ce::argus_server: 'cctbargus01.in2p3.fr'
htcondor_ce::argus_port: 8154
htcondor_ce::argus_resourceid: 'http://cc.in2p3.fr/ce'
htcondor_ce::use_static_shadow: false
htcondor_ce::job_router_entries: >-
   [ \
     eval_set_environment = debug(strcat("HOME=/tmp CONDORCE_COLLECTOR_HOST=", CondorCECollectorHost, " ", \
     TargetUniverse = 5; \
     name = "Local_Condor"; \
     set_VOName = ifThenElse(isUndefined(X509UserProxyVOName),"LOCAL",X509UserProxyVOName); \
     set_AcctSubGroup = toUpper(\n       ..... 
       eval_set_RequestMemory = ifThenElse(WantWholeNode is true, !isUndefined(TotalMemory) ? TotalMemory*95/100 : JobMemory, OriginalMemory); \
     ]
   ]
site_htcondor_ce::prerelease_repo_enabled: false
# BDII
htcondor_ce::install_bdii: true
htcondor_ce::supported_vos:
   - atlas
   - ..... 
htcondor_ce::goc_site_name: 'IN2P3-CC'
htcondor_ce::benchmark_result: '10.26-HEP-SPEC06'
htcondor_ce::execution_env_cores: 40
Job routers in local htcondor and htcondor-CE

Hierarchy tree is based on VO names and user proxy role.

- `group_u_VONAME`
  - `group_u_VONAME.grid_{VONAME}{ROLE}`

Assuming that each proxy role represents a different activity, but a VO can use the same role for different activities.
Until now …
LDAP user is authenticated using GSI authorization

```
55 * * * * ldap voms-proxy-init --cert /var/lib/ldap/hostcert.pem --key /var/lib/ldap/hostkey.pem --valid 02:00
```

To check the publication

```
$ ldapsearch -LLL -x -H ldap://cccondorcm01.in2p3.fr:2170 -b o=glue
```

Note: The resource must be added into site BDII configuration to be published

```
$ vi /etc/bdii/gip/site-urls-CC.conf
....
CONDOR01 ldap://cccondorcm01.in2p3.fr:2170/mds-vo-name=resource,o=grid
```
In particular, I would like to know if the number provided by cccondorcm01 is correct to take into account for the total cores provided by your site.

Changes in the /var/lib/bdii/gip/provider/htcondor-ce-provider file:

```plaintext
vcpus = 0 (line 140)
vcpus=sum(total_cores.values()) (line 163)
cpu=tup[5] => cpu=vcpus (line 179)

arch=tup[0].lower()
os=tup[1].lower(),
```

$ ldapsearch -LLL -x -H ldap://cccondorcm01.in2p3.fr:2170 -b o=glue | grep CPU
GLUE2ComputingManagerTotalLogicalCPUs: 2680
GLUE2ExecutionEnvironmentLogicalCPUs: 2680
GLUE2ExecutionEnvironmentCPUMultiplicity: singlecpu-multicore
Add the resource to your argus policies

```yaml
resource "http://cc.in2p3.fr/ce" {
    obligation "http://glite.org/xacml/obligation/local-environment-map" {
    }

    action ".\.*" {
        rule permit { pfqan="/atlas/Role=pilot" }
        rule permit { pfqan="/atlas/Role=lcgadmin" }
        rule permit { pfqan="/atlas/Role=production" }
        rule permit { pfqan="/atlas/Role=software" }
        ...
    }
}
```

The grid-mapfile is static in argus server, for historical reasons specific roles could be matched only to one account or to an account of the VO pool:

```
"/cms/Role=lcgadmin" cmsgrid
"/cms/Role=pilot" .cms
```
Using `condor_history` command to generate json files by hour and by schedd and to save into a database.
- Testing Apache Spark
- Tableau – Data visualization tool

Adapting our local scripts to generate APEL accounting files.
- Fifemon looked like the way to go
  - But CC-IN2P3 doesn’t have graphite in its monitoring stack
  - Adapting it to Collectd + Elastic Search was time consuming and quite unsuccessful.

- Collectd + dedicated plugin
  - Currently writing a Python plugin for HTCondor
  - Using Python Bindings to gather classads
    - Though the Collector
      - Negotiator
      - Collector
      - Defrag
      - Schedd
      - Startd
    - Through the Schedd
      - Schedd ads (jobads of all jobs, regardless of their state)
Collectd + plugin for HTCondor-CE as well
  ◦ Query the HTCondor-CE collector with a simple trick

Import htcondor
ce_coll = htcondor.Collector('localhost:9619')
ce_coll.query()

Changing the port you get connected to CE

◦ We gather
  • Job general stats
  • Schedd daemon metrics

◦ Currently putting efforts in writing a proper reusable code for public release
Grafana to plot the metrics
HTCondor Configuration
Job Resource Requests

- CPU’s
- Memory
- Wallclock
- Disk (N/A)

Max limits

- +xcount: Max 8 slots.
- +maxMemory: Max 4GB by core.
- +maxWallTime: Max 96 hours.

The default values are set according to the VO demands and internal capacity. (CPUs vs Memory consumption)
Accounting is based on groups depending on x509 user proxy role.

There is not separation between single core and multicores.

Let the VOs to used the “allocated” resources as they wish.

◦ Local capacity plan will be enforce (respect the allocation of slots ~100%).
◦ (e.g. the maximum number of high memory jobs >= 4Gbytes per core)
Each accounting group / subgroup has a concurrency limit defined

**Job Router**
- `set_ConcurrencyLimits = \strcat(toUpper(X509UserProxyVOName),",",AcctSubGroup,":\",MY.RequestCpus);`

**Negotiator**
- CMSLCG_LIMIT=10
- ATLASLCG_LIMIT=10
- ...

This helps to control the number of running job by group.
Dynamic Quota and “Surplus“ for all VOs
- Share tree policy with groups and subgroup for optimum utilization of resources.

Group Rebased Policy
- in order all VO to be “delighted” from over-pledges resources.

Trying to avoid practices which disturbs or constrain the Fairshare Scheduling
- Reshape the multicore jobs priorities in order to facilitate the starting of the jobs.

It could be special priorities for special users
- SAM tests

Try to use “breath-in” ramp-up mode of the machines
- Uniform job distribution across all the machines
Single core (1) and multicore jobs (8-cores) should running simultaneously on a machine.

- The problem of fragmentation of resources is that over time the machine resources may become partitioned into slots suitable only for running single core jobs.

- If eight (8) single slots do not happen to become idle at the same time on a machine, then a multicore jobs with 8 cores will not be able to start that machine. Even if the multicore job has a better priority than the single jobs
Multicore job (8 cores) is not fit as just one S-core job claims 1 slot for the last “pocket” of 8 cores.

Therefore:

1. **Drain** this done in order to get back 8 core jobs idle at same time and reduce the utilization for a while.

2. **Dispatch more single core jobs** in order to get 100% the machine

The fragmentation issue became significant for the last 8 pocket slots.

Note: Memory is no being taking into account.
Defrag process

- Estimating the expected maximum number of multicore jobs on Steady state → mc target

- Define the difference of number of multicore jobs on Steady state minus (-) the current one in term of 8 slots “pockets”

- Start the defrag daemon when the GAP is positive (GAP >0):
  - Perform partial drain up to 8 slots:
    - \( \text{WHOLE\_MACHINE\_EXP} = (\text{partitionable} \&\& \text{cpus} \leq 8) \)
    - \( \text{DEFRAG\_MAX\_WHOLE\_MACHINES} = \text{DEFRAG\_MAX\_CONCURRENT\_DRAINING} \) could be maximum to the 50% of the number of machines
    - \( \text{DEFRAG\_MAX\_CONCURRENT\_DRAINING} = \min (\text{GAP, max # of machines} / 2) \)
    - \( \text{DEFRAG\_RANK} \sim \text{Number of running single core jobs per machine} \)
    - \( \text{DEFRAG\_DRAINING\_MACHINES\_PER\_HOUR} \) could be arbitrary high (e.g. 4 times the GAP) just to start the process as soon as possible

  - We reduce linearly the parameters (\( \text{MAX\_WHOLE\_MACHINES}\) \( \text{MAX\_CONCURRENT\_DRAINING} \)) according to the GAP values.

- Stop the defrag when the GAP is negative (GAP <0)

- Use an external cront script in order to “feed” the parameters of the defrag daemon.
  - Debrag interval is 300 sec and external script interval is 600 sec
  - The values are indicative for debug
According to VO demands the mc target should be 55% of the total slots (2680) at given moment.

Difficult to attain the MC Target (~1474 slots) in a period of drain ~ 5 days (?)

The increase in score job is due to an incident of mc job submission clients on 21 Sep 2019.

But we stop the drain in order to prepare the system for another test.
• Attain maximum utilization ~98% but is not sustainable. This is probably due to the shorts wallclock time of mc jobs (to be checked)

• This metrics represent the # for free slots for machine where the TotalSlots-8 are already filled.

• This metrics is a direct measurement of the non used slots in an almost full machine due to draining or due to fragmentation.
A lot of things to do!! Just starting

- A short term:
  - Verify that accounting values are correct next week.
  - Migrate all grid resources before February 2020.
  - Publish Collectd python plugin for HTCondor-CE and HTCondor ASAP.
  - Continue improving our configuration.
Is easy to deploy HTCondor / HTCondor-CE using puppet modules.

Defrag daemon is not aware about running condition on the farm therefore
- We need an external script to feed the defrag parameter according to the runtime conditions on the farm (e.g. start or stop the defrag process)

Needs to understand better the “wallclock” time distribution of single core jobs vs the multicore jobs.
- Just one single core job could delay the total delivery of 8 core pocket (assume that the other 7 are drained earlier)
A big THANK YOU!!!

To HTCondor developers, specially to:
- Miron Livny
- Todd Tannenbaum
- John (TJ) Knoeller
- Brian Lin
- Zachary Miller

CC-IN2P3 colleagues
- Cécile Evesque
- Ghita Rahal