Migrating to a new job scheduling system on a grid computing cluster

Bachelor thesis of
Michael Margos
RWTH Aachen University
Physics Institute III B

Referees:
Prof. Dr. Achim Stahl
Prof. Dr. Christopher Wiebusch

December 28, 2018

Contents

1 Introduction

The grid cluster of physics institutes of RWTH Aachen University has an outdated job scheduler, which does not handle large amounts of jobs well and poses a security risk. To solve this issue, an up-to-date job scheduler is chosen, installed on two test computers, configured to have the same functionality as the previous job scheduler, its features and robustness are tested, and results are compared to the previous job scheduler. A list of possible future improvements is given at the end.

2 Grid Computing Cluster

The computing grid is a set of connected computing clusters over multiple locations that perform computations and store and process data. These computers reside in different administrative domains, but are made available for specific tasks using grid middleware, which restricts access to computing resources—such as CPU time, RAM and data storage—to authorized users.

A computing cluster is a set of locally connected computers or computing resources that act as a singular powerful computer. They are split into a single master node and multiple worker nodes. Computation tasks received at the master node are distributed to worker nodes using a batch system. A computing cluster may also be part of a computing grid, e.g. the Worldwide LHC Computing Grid [1]. Computing clusters usually have uniform software environment and access to a shared data storage, so computing processes may be scheduled to run on any of the worker nodes.

The physics institutes I B, III A, and III B of RWTH Aachen University are participating in the CMS experiment at CERN. To access data and analyze it, their computing cluster is part of the WLCG and provide a joint Tier 2/Tier 3 grid cluster [2]. Tier 0 is the CERN Data Center, which stores experimental data, reconstructs some of it and distributes it to Tier 1 centers. Tier 1 computer centers are responsible for storing raw and reconstructed experimental data, reconstructing most of the data and distributing them to Tier 2 centers. Tier 2 clusters are available to all CMS scientists, while Tier 3 clusters are normally limited to the local institute; in our case, all German CMS scientists and the astroparticle groups of institutes I B, III A, and III B may use this cluster as well.

The computing cluster currently uses the CREAM [3] job management interface on the Unified Middleware Distribution [4]. The batch system responsible for job execution, resource allocation and enforcing group quotas are the resource manager TORQUE [5] and the job scheduler Maui [6]. Scientific Linux 6 is used as the operating system.

Scientists involved in the CMS experiment are part of the CMS virtual organization (VO). Other virtual organizations that have access to resources of the grid cluster are DCMS, Pierre Auger Observatorium, and Icecube. Those VOs are mapped to Unix groups, allowing simple file access rights definition.

The grid cluster in use is located in twelve computer racks at the IT Center of RWTH Aachen University and consists of a computer acting as the computing element and 262 worker nodes with a total of 5488 cores. It is connected to the dCache storage system, which holds 3.8 PiB data. [7]

3 Batch System

Batch processing systems queue, schedule and execute batch jobs. These jobs are computer programs that do not need user interaction. Batch systems are particularly useful in natural sciences, where a lot of computation is needed and large data need to be processed. Batch systems usually consist of a central manager, one or more submit machines, and multiple execution machines. The central manager gets prepared job submissions from submit machines, puts them in a queue, and assigns them executing machines (resources) based on job scheduling policies. Their usage and functionality varies from lightweight single-machine systems with first-in-first-out or time-based job scheduling to high performance computing cluster batch systems with a lot of features like user authentication, detailed job monitoring, multiple queues, and group-based quotas.

An important aspect of batch systems is scalability. As batch systems need to manage multiple jobs from different submitters, the addition of jobs to the queue, and manage multiple resources, batch systems can reach a limit of how many execution machines and how many jobs it can handle.

Typical batch jobs submitted by high energy particle physicists include Monte Carlo simulation or data analysis. These jobs have the useful property of being trivially parallelized, as analyzed or simulated data can be split into smaller batches.

3.1 Job Schedulers

The resource manager TORQUE and the job scheduler Maui, which are currently used to manage jobs on the grid cluster, cause problems and have to be replaced. Neither TORQUE nor Maui are actively developed or updated, so programming bugs will not be fixed, which leads to a potentially vulnerable system. The communication between them is inefficient, so that Maui requested a list of queued and running jobs from TORQUE nearly as frequently as TORQUE could respond, when around more than 5000 jobs were queued. Because of that, a new job scheduler, which will also be used as the resource manager, has to be chosen. The choice is limited by the job schedulers CREAM is able to communicate with: Portable Batch System (PBS; TORQUE is a variant of PBS), Load Sharing Facility (LSF), Sun Grid Engine (SGE), Slurm Workload Manager (Slurm), and High-Throughput Condor (HTCondor).

PBS was developed at NASA since 1991 [8] and has an open source version called OpenPBS, which TORQUE improved upon. Both versions are outdated (last software patch for OpenPBS was published in January 2004 [11], and the company Adaptive Computing ceased updates for TORQUE in June 2018 [5]). There is an actively developed variant called PBS Professional, either as professional support with a commercial license [9] or available as open source [10].

LSF is a job scheduler originally released by Platform Computing (now owned by IBM), and has an open source version called OpenLava, but is commercially available with more functionality and professional support by IBM as IBM Spectrum LSF. [12]

SGE was a job scheduler by Sun Microsystems, then renamed to Oracle Grid Engine when Oracle took over Sun Microsystems, then renamed to Univa Grid Engine after Oracle sold the intellectual property to Univa Corporation. "Open source forks are Son of Grid Engine and Open Grid Scheduler" [13].

Slurm is an open source job scheduler for Linux operating systems and is maintained by SchedMD. [14]

HTCondor is also an open source job scheduler, developed at the University of Wisconsin-Madison. [16]

From the given options, the choice was made to use HTCondor. The general reasons are: HTCondor is up-to-date, is open source software and has good scalability. The more practical reasons behind this choice is the prior knowledge about and experience with of HTCondor, since it is already deployed on the office computers in the physics institutes I B, III A and III B of RWTH Aachen University, and thus assures that HTCondor works as a resource manager and job scheduler. Other researchers—especially those involved with CMS—also are going to switch to or already use HTCondor, so a future change from CREAM to HTCondor ordered by the CMS grid management seems possible, thus further enforcing HTCondor as the job scheduling software of choice.

3.2 Current Configuration

In order to replicate TORQUE's and Maui's behavior within the new job scheduling system, it is important to look at their current configurations.

3.2.1 TORQUE Configuration

TORQUE's configuration includes multiple queues that are restricted to the groups that can be assigned to the queue, number of jobs that are running or queued, and their total CPU time and real time (walltime). Jobs in the queues `cms`, `auger` and `icecube` are restricted to just over two days of CPU time and three days of walltime, while jobs in the queues `ops` and `dteam` are limited to only just over two hours of CPU time and four hours of walltime. This is done to have a safety measure against jobs using too much time. Processes do not use up CPU time, if for example the need for a file to be loaded from disk into memory to work on. This is the reason for the walltime setting being higher than the allowed CPU time.

queue	acl_groups	max_queuable	max_running	CPU time	walltime
cms	cms, dcms	16464	5488	49:00:00	72:00:00
auger	auger	8232	5488	49:00:00	72:00:00
icecube	icecube	8232	5488	49:00:00	72:00:00
ops	ops	20	4	02:10:00	04:00:00
dteam	dteam	20	4	02:10:00	04:00:00

Figure 1: configuration values for queues in TORQUE

3.2.2 Maui Configuration

Of importance are fair-share quotas, as set in Maui's configuration file. Maui uses a fair-share configuration that is spread over three layers: account, QOS, and groups. The assignment of a job to a slot on an executable machine is then made based on the job, with its associated account being prioritized over its associated QOS, and QOS being prioritized over group and user.

Three account configurations are set, `cms-all`, `astro`, and `ops-dteam-all`, with their fair-share target set to 97.12%, 2.88% and 0.01% of all available resources respectively (see figure 2). Although this adds up to slightly over 100%, this configuration seems to work. Additionally, the amount of jobs that are allowed to run are restricted for `ops-dteam-all`.

```
ACCOUNTCFG[cms-all]        FSTARGET=97.11828741948355
ACCOUNTCFG[astro]          FSTARGET= 2.8817125805164516
ACCOUNTCFG[ops-dteam-all] FSTARGET= 0.01     MAXJOB=4,40
```

Figure 2: accounts in Maui's configuration

Furthermore, a subdivision of accounts (called quality of service (QOS)) is configured with fair-share targets of 33.71% for `cms-user`, 63.41% for `dcms-user`, and 1.44% for `auger-all` and `icecube-all` (see figure 3).

```
QOSCFG[cms-user]           FSTARGET=33.70963865646851
QOSCFG[dcms-user]          FSTARGET=63.40864876301504
QOSCFG[auger-all]          FSTARGET= 1.4408562902582258
QOSCFG[icecube-all]        FSTARGET= 1.4408562902582258
```

Figure 3: subdivision of accounts in Maui's configuration

In figure 4, the different user groups are then assigned to those previously defined accounts, most of them to a QOS, and some are given a higher priority that enables their jobs to be preferred.

```
GROUPCFG[dteam]   ADEF=ops-dteam-all    PRIORITY=1000
GROUPCFG[ops]     ADEF=ops-dteam-all    PRIORITY=1000
GROUPCFG[cms]     ADEF=cms-all          QDEF=cms-user
GROUPCFG[dcms]    ADEF=cms-all          QDEF=dcms-user
GROUPCFG[auger]   ADEF=astro            QDEF=auger-all
GROUPCFG[icecube] ADEF=astro            QDEF=icecube-all
```

Figure 4: groups in Maui's configuration

This allows for a hierarchical grouping of quotas, by distinguishing between different account fair-share quotas, and then dividing them into subdivisions with their own set of fair-share quotas.

Some groups of users have got a higher priority, and additional resources are reserved for some groups, individual users, or jobs submitted to a short queue.

```
SRCFG[ops_dteam] TASKCOUNT=1 RESOURCES=PROCS:1
SRCFG[ops_dteam] HOSTLIST=grid-wn00[1-2].physik.rwth-aachen.
    de,grid-wn10[1-2].physik.rwth-aachen.de
SRCFG[ops_dteam] CLASSLIST=ops,dteam
```

Figure 5: slots reserved for prioritized groups (here shown for `dteam` and `ops`)

An example for reserved resources (in this case processors) for the groups `dteam` and `ops` is given in figure 5.

Maui is configured to have a fair-share decay, which lets users that have not recently used the resources to be prioritized over others in the same group. The recent resource usage is weighted with a decay factor of 0.9^N for every time window N of length 48 hours in the past, up to 8 ($N = 7$) time windows [15]. Within a fair-share group, the user with the smallest weighted recent usage will be queued first. The values for the decay factor and time window are set as shown in figure 6.

```
FSINTERVAL      48:00:00
FSDECAY         0.9
```

Figure 6: fair-share decay

4 HTCondor

HTCondor [16] is open source software developed by the Center for High Throughput Computing at the University of Wisconsin-Madison [17]. It is used to distribute jobs across different computers (executing machines) to enable High Throughput Computing (HTC). The original use case is to use already available personal computers and workstations, that would otherwise stay idle, for computing intensive processes. This is the case in research institutes, where computing intensive simulation and analysis is needed, but access to high performance computing is expensive or otherwise limited. Despite its initial use case and because of its useful functionality, HTCondor can be utilized as a resource manager and job scheduler for a computing cluster.

4.1 How HTCondor Works

Multiple machines within an organization like an institute or a company are grouped together in the same HTCondor pool. Each pool has one central manager that is responsible for collecting and distributing job submissions. When submitting a job, the user creates a submission file, which includes the file path to the executable, the HTCondor execution environment (the "universe"), and a lot of optional features, like input/output files, command line arguments, environment variables, job requirements and machine preferences. The user then submits the job with `condor_submit [submission file]`, causing HTCondor's scheduler to transform the submission file into a detailed ClassAd, which is sent

to the pool's collector (`condor_collector`; see step 1 in figure 7). The collector gives ClassAds from all job submissions and all execution machines to the negotiator (`condor_negotiator`), that uses them to assign jobs to slots on executing machines (step 2 in figure 7) based on user priorities, availability, machine and job requirements, as well as machine ranking. After an execution machine receives a job, it retrieves the program to run—the executable—and input files from the submit machine or, when available and configured, from a shared file system (step 3 in figure 7).

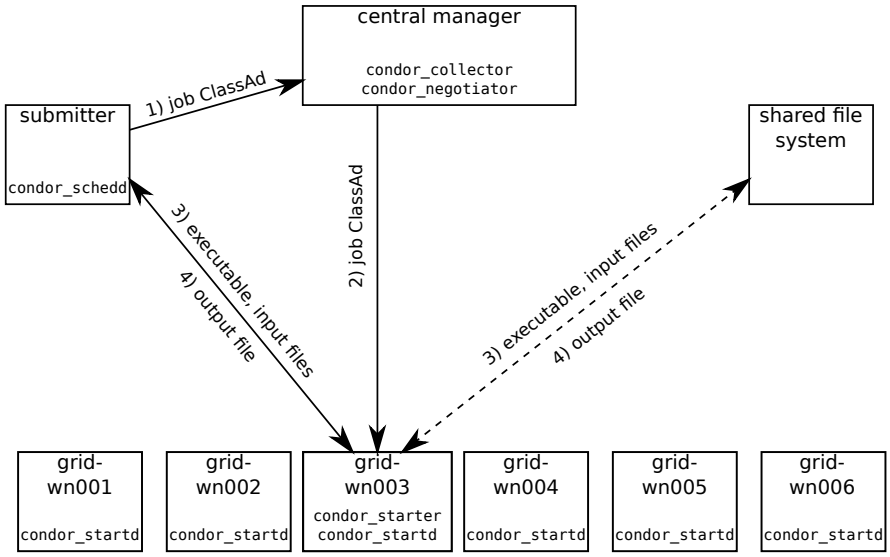

Figure 7: job submission, data transfer, and condor daemons

4.2 ClassAds

ClassAds are the basis of HTCondor. Each slot of an execution machine and each job are represented and thus "advertised" as a ClassAd. They contain information, which is used to match jobs to execution slots. ClassAds are basically a list of attribute-value pairs, where the values are made of literals (strings, numbers, `undefined`, or `error`), operators (elementary arithmetic operations and comparison operators), references to other attributes within its own ClassAd or its matched ClassAd, and built-in functions. "[T]he keywords `TRUE` and `FALSE` (case insensitive) are syntactic representations of the integers 1 and 0 respectively". Attribute names may consist of letters, underscores and digits, but may not begin with a digit. The names are case insensitive. [19]

A short example ClassAd for an execution slot is shown in figure 8. In this example, some general information about the machine slot is given, as well as its available resources,[1] its state, `Requirements` for accepting new jobs, a

[1]The units are inconsistent for data capacity: disk space is given in kiB, but memory is

```
Activity = "Idle"
Cpus = 4
Disk = 115943020
KeyboardIdle = 5
LoadAvg = 0.1
Memory = 6324
Name = "slot1@lx3b39.physik.rwth-aachen.de"
OpSys = "LINUX"
Rank = 0
Requirements = (START)
Start = (KeyboardIdle > 15*60) && (LoadAvg <= 0.3)
State = "Owner"
```

Figure 8: a short example for a ClassAd displaying a slot of an execution machine

`Rank` (preference) for specific jobs, and the `Start` expression. `Requirements`, `Rank` and `Start` play an important role in negotiation and will be discussed in subsection 4.6. The startd daemon periodically creates a ClassAd for the machine, depicting its current state, and submits it to the central manager's collector daemon.

Every job has a job ClassAd that is used by the negotiator to match against machine slot ClassAds. In figure 9, a truncated ClassAd for three jobs submitted through submission file 10 is shown, that only differ in their arguments and output files. The jobs require a machine with the given operating system (in this case: Linux) and enough Memory. `RequestMemory` is set by the result of the built-in function `ifthenelse`, so if `MemoryUsage` is `defined`, then `RequestMemory = MemoryUsage`, else `RequestMemory = (ImageSize + 1023) / 1024)`. Division by 1024 is needed, as the ImageSize is given in KiB, but Memory is given in MiB. Also, there is implicit integer division, which means that the result of the division is itself an integer number. Any division remainder will be discarded. The ClassAd may be updated during the jobs' runtime to show the current status or to update the correct image size.

4.3 Submission Files

Job submission files play an important role in job management, since they give all information necessary to run a job. They consist of attribute-value pairs like ClassAds. There are a few pre-defined attributes (like `ProcId` in figure 10). The only mandatory attribute is `Executable`, which gives the path to the program to run. `Universe` sets the HTCondor execution environment, and will always be set to "vanilla" by CREAM. Jobs running in the vanilla universe have nearly no restrictions, but they also do not profit from HTCondor's features like job checkpoints, which would allow a job to make a checkpoint and to be stopped on one executing machine, so it can continue running from the last checkpoint on another executing machine. `Input` and `Output` are file replacements for the executable's console input, output; `log` defines the file, where HTCondor shall write the job's status; `Args` are the arguments given to the executable; `queue`

given in MiB.

```
ClusterId=1
Cmd="/home/cms001/testjob.sh"
JobUniverse=5
Owner="cms001"
AcctGroup = "group_cms.cms"
UserLog="/home/cms001/testjob.log"
Requirements=(TARGET.OpSys == "LINUX") && (TARGET.Memory >=
    RequestMemory)
RequestMemory=ifthenelse(MemoryUsage =!= undefined,
    MemoryUsage, (ImageSize + 1023) / 1024)
ImageSize=1
Rank=0.0
custom_ClassAd_attr=true
ProcId=0
Args="0"
JobStatus=1
Out="00.out"

ProcId=1
Args="1"
JobStatus=1
Out="01.out"

ProcId=2
Args="2"
JobStatus=1
Out="02.out"
```

Figure 9: an example for a job ClassAd

defines the number of jobs sent with this submission file, but gives them increasing `Process` values, starting from 0; `accounting_group` will be explained in section 5.2. A few functions can be used while defining values inside submission files. The example uses `$INT`, which takes the first argument in parenthesis and turns it into a character string according to the format specified in the second argument. There are other functions available, but they are not needed by CREAM nor are they used for testing.

When a user (or program) submits jobs via `condor_submit [submission file]`, a job ClassAd is made, taking information from the submission file, and further attributes are added, like the user name of the submitter and the machine where this submission is sent from. Any arbitrary attribute-value line is taken from the submission file and inserted directly into the resulting job ClassAd, if it begins with a plus sign (+). In this example, the line `custom_ClassAd_attr = true` will be inserted into the job ClassAd (as shown in figure 9).

4.4 Configuration Files

HTCondor has a central configuration file, which is used by every HTCondor daemon and is usually located at `/etc/condor/condor_config`. It is read every time HTCondor is started or the command `condor_reconfig` is issued, and

```
Executable      = testjob.sh
Universe        = vanilla
Args            = $(ProcId)

output  = $INT(ProcId,%02d).out
log     = testjob.log
accounting_group = group_cms.cms
+custom_ClassAd_attr = true

queue 3
```

Figure 10: a submission file used for `condor_submit`

contains attribute-value pairs, which are used to enable, disable, or further specify some features. Most of the attributes have useful default values when not explicitly set, but there are a few required configuration attributes (also called macros[2]) without default values. Like ClassAds, attributes can reference other attributes and have values made from literals and operators, but can have different values for different daemons. The same built-in functions as in submission files may be used.

```
LOCAL_CONFIG_FILE = /etc/condor/condor_config.local
LOCAL_CONFIG_DIR  = /etc/condor/config.d
CONDOR_HOST  = grid-tmp5.physik.rwth-aachen.de
CONDOR_ADMIN = condor@physik.rwth-aachen.de
COLLECTOR_HOST  = $(CONDOR_HOST)
ALLOW_READ  = *.physik.rwth-aachen.de
ALLOW_WRITE = *.physik.rwth-aachen.de
DAEMON_LIST = MASTER COLLECTOR NEGOTIATOR SCHEDD STARTD
START = (KeyboardIdle > 15 * $(MINUTE)) && \
        (LoadAvg <= 0.3)
MINUTE = 60
```

Figure 11: sample configuration file

Figure 11 shows a sample configuration file. It contains references to a local configuration file and a directory holding zero or more files that have additional configuration options or can even overwrite macros. The `CONDOR_ADMIN` contains the email address HTCondor mails to in case of a failure (e.g. a daemon crashes). The `COLLECTOR_HOST` points at the central manager. `ALLOW_READ` and `ALLOW_WRITE` restrict permissions to domain names or IP ranges. The `DAEMON_LIST` tells the `condor_master` which daemons to start. The `START` expression will be automatically included in the machine's ClassAd requirements, and is therefore used to control when and which jobs the machine should run.

[2]A nearly full list of configuration macros is available at [20].

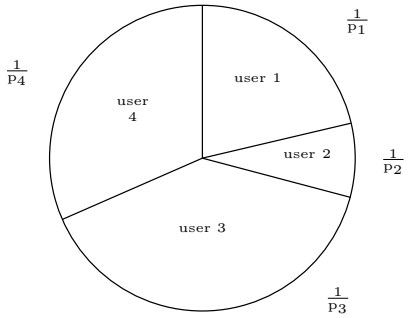

Figure 12: pie chart

4.5 Daemons

HTCondor uses background processes, so-called daemons, to maintain functionality. The most important daemons are:

`condor_master` starts all other daemons as set in the configuration file. Every machine in the pool has this daemon.

`condor_collector` collects all job and execution machine ClassAds, and gives them to the `condor_negotiator`. It runs on the central manager.

`condor_negotiator` matches jobs to machines. It runs on the central manager.

`condor_schedd` sends job ClassAds and claims matched machines. It runs on submit machines.

`condor_startd` sends machine ClassAds and enforces execution policies. It runs on execution machines.

`condor_starter` monitors a job and sends information to the submit machine. It is started on the execution machine as soon as a job is started.

There are more daemons[3], but these are the daemons doing the main work.

4.6 User Priorities and Negotiation

The `condor_negotiator` is responsible for assigning jobs to execution machines. Each user gets a portion of execution machine slots—a pie slice—proportional to the inverse of the user's priority. Jobs are assigned to machines in user priority order, beginning with the smallest priority, until the user's pie slice is filled or until the user has no more jobs. After all users have been dealt with, another negotiation cycle begins, until there are no more jobs or no more idle execution machines available.

For each job, every machine ClassAd is compared to the job ClassAd to see if machine requirements and job requirements are fulfilled, and all machines that fulfill this condition are then sorted by job rank, jobless machines, machine rank, and better user priority than the currently running job. The job is then assigned to the first machine in this sorting list.

[3] A full list of daemons and their description is available at [18].

5 HTCondor on a Grid Cluster

A pair of machines with similar hardware and same operating system as on the grid computing cluster was prepared to install, configure and test HTCondor without affecting current cluster usage. The machines used in the test environment are `grid-tmp5`, which shall take the role of the central manager and job submit machine, and `grid-wn048`, which shall act as the executing machine (worker node).

HTCondor was installed on those machines in the newest stable version 8.6.12[4] by the grid manager using the software deployment system Quattor[21]. While CREAM already has scripts that help to link it to HTCondor, Quattor has pre-made shell scripts and configuration files, which assist in connecting CREAM and HTCondor and contain configuration options used in some grid clusters. These configurations still need manual editing to fit the grid clusters needs.

5.1 Security Settings

User authentication is made through X.509 Certificates to the grid middleware UMD. HTCondor has its own options for user authentication, authorization, and network integrity [23], but the middleware already takes care of user authentication and authorization, so HTCondor needs to be configured to allow only the grid's central manager machine and the worker nodes to communicate with each other. This is done using a host-based security model. On both the central manager and the execution machine configuration files the configuration template `use SECURITY : Host_Based` is used, as shown in figure 13. It sets some security related configuration macros to useful default values. The name of these macros all begin with `ALLOW_`, and their values can be overwritten by reassigning the macro after the line where the configuration template was used. This is used to reassign `ALLOW_READ` from `*` (everyone) to `grid-*.physik.rwth-aachen.de`, meaning only (users on) machines from the `physik.rwth-aachen.de` domain with host names beginning with `grid-` are allowed to read information about the pool and thus use `condor_status` and `condor_q`.

```
use SECURITY : HOST_BASED
ALLOW_READ   = grid-*.physik.rwth-aachen.de
ALLOW_DAEMON = condor_pool@physik.rwth-aachen.de

# This is to enforce password authentication
SEC_DAEMON_AUTHENTICATION = required
SEC_DAEMON_AUTHENTICATION_METHODS = password
SEC_CLIENT_AUTHENTICATION_METHODS = password,fs,gsi
SEC_PASSWORD_FILE = /var/lib/condor/condor_credential
TRUST_UID_DOMAIN = false
```

Figure 13: allowing only the institute's grid machines to communicate with each other

[4]During the course of this work, HTCondor was automatically updated to version 8.6.13, which was released on October 31st, 2018.

Another security mechanism is the use of daemon authentication. Quattor has set password authentication for daemon communication, and this setting was adapted. The current grid manager already generated the password and saved the hashed password into `condor_credential`. Quattor's configuration option allows daemon communication only the user `condor_pool`, and this is further constrained to have this user come from the `physik.rwth-aachen.de` UID (user identifier) domain. Additionally, since Quattor's configuration option enabled blindly trusting the UID domain, the `TRUST_UID_DOMAIN` was set to false to re-enable UID checking.

5.2 Fair-share Scheduling

In order to allow every organization or institute to utilize the grid cluster proportional to a previously defined quota, a quota based fair-share scheduling will be needed. HTCondor's standard method of determining a fair use of available resources to all users is through user priorities. This method assumes that all users have equal right to use the machine pool, that this right is independent of other users from the same organization/institute, and that machine allocation should be decreased for users who recently made heavy use of the machine pool. To enforce a group based quota for fair-share scheduling, three options will be discussed: adjusting user priority factors, changing machine ranks, and HTCondor's built-in group accounting mechanism.

This grid cluster has additional restrictions. Most jobs that run on the grid are long running (around eight hours to two days) and cannot be checkpointed, so the usual method of preempting a machine to give it a higher priority job is bound to loss of hours or potentially days of computing time and therefore discouraged.

Another restriction is the need to give some jobs preferential treatment. These jobs are either test jobs by CMS or the testing virtual organization OPS to check availability and functionality, or jobs submitted to the short queue, which in return are limited in their runtime.

priority factor: There are two types of user priorities: real user priority (RUP) and effective user priority (EUP) [24]. The RUP is a number equal to the user's recent usage of the pool, and the EUP is just the RUP times a user priority factor. In each matchmaking cycle, each user gets free slots proportional to the inverse of his EUP. While the `DEFAULT_PRIO_FACTOR` is set to 1000.0, it can be individually set for each user by using the command `condor_userprio -setfactor [user] [user priority factor]`. If we want the slots used to be proportional to a quota, but slots are assigned proportional to the inverse EUP, then we need to set the `user priority factor` proportional to the inverse square of the quota, as indicated in [26]

There are two problems that arise with this configuration: first, the needed user priority factor is dependent on other user priority factors; and second, those user priorities are only for single users, not for groups. The quota-dependent user priority factor would need to be set for every user of a group, and negotiation would need to assign slots based on all group's total EUPs and not the user specific EUPs. While this approach to solve

group-based fair-share is doable, another approach might be easier and more exact.

machine rank: Execution machines can be configured to prioritize jobs based on information given in the job ClassAd. The configuration attribute used for this is `RANK`. The higher the machine rank expression evaluates, the more the machine prefers a given job. This can be utilized to prioritize specific users or user groups. Assigning a number of machines to groups according to their quota could mimic such a quota-based scheduling. User priorities are still in effect, but the rank has higher precedence over user priority. Unfortunately, machine ranks are only considered when machine preemption is enabled, and preemption is not desired and therefore turned off on the grid cluster. This means that enforcing quota-based fair-share scheduling by editing machine ranks is not an option in this case.

group accounting: HTCondor has a built-in configuration option that allows enforcing quotas on jobs from different groups. This option allows groups to use their share of machines. It is also possible to use other group's idle machines, when both groups are allowed to accept surplus. The group's share of resources can only be instantly granted when they either deny surplus, reserving their share for their personal use, or when preemption is enabled. Jobs need to wait in the queue for a slot to accept jobs (i.e. another job has finished) before its group's fair share can be accounted for. Test jobs and the short queue can be put into a group with disabled surplus share, but that would lead to a portion of the resources not utilized unless there are test jobs or short queue jobs to run.

The option of group accounting was chosen as the method that fits best to fair-share scheduling needs. The issue of giving certain jobs preferential treatment is solved by reserving a slot per machine to this kind of job only.

5.3 Group Accounting

Group Accounting is configured by declaring groups and then defining quotas for those groups in the configuration file read by the negotiator, i.e. in the central manager's configuration file. Figure 14 shows the group declarations. Job ClassAds need to specify the group they belong to, otherwise their jobs will be treated as belonging to the group `<none>`. Jobs from the group `<none>` are the last to be considered within a negotiation cycle, if there are still idle execution machine slots available. The groups `ops` and `dteam` are not transferred from Maui's configuration to HTCondor's configuration, because their jobs shall not be subject to group accounting quotas, but instead be treated as high-priority jobs.

```
GROUP_NAMES = group_cms, group_cms.cms, group_cms.dcms, \
              group_auger, group_icecube
GROUP_ACCEPT_SURPLUS = true
GROUP_AUTOREGROUP = false
```

Figure 14: group accounting settings

GROUP_AUTOREGROUP, if set to true, would allow jobs to be re-negotiated as belonging to the group <none>, if their groups quota is already used. There is no need to enable automatic regrouping. Since the groups shall be allowed to go over their quotas, GROUP_ACCEPT_SURPLUS is set to true.

```
GROUP_QUOTA_DYNAMIC_group_cms       = 0.9711828741948355
GROUP_QUOTA_DYNAMIC_group_cms.cms   = 0.347098775649
GROUP_QUOTA_DYNAMIC_group_cms.dcms  = 0.652901224351
GROUP_QUOTA_DYNAMIC_group_auger     = 0.014408562902582258
GROUP_QUOTA_DYNAMIC_group_icecube   = 0.014408562902582258
```

Figure 15: group quotas

Assigning quotas to groups is done by taking values from Maui's configuration file, dividing them by 100%, and setting values for GROUP_QUOTA_DYNAMIC accordingly, as shown in figure 15. Only exception to this are subgroups, in this case group_cms.cms and group_cms.dcms (subgroups of group_cms), which have their quota defined in relation to their supergroup. HTCondor will recalculate group quotas, should the values add up to something other than 1. For testing purposes, the correct quotas are commented out and replaced by more practical values (see figure 20 in section 6.1).

5.4 Assigning Groups

Job submission files need to contain the accounting group's name, as they are defined in the configuration file. CREAM uses submission scripts to generate a HTCondor submission file. At some point it uses an XML file containing regular expressions to determine the job's group and policy by matching an identity string containing information about the virtual organization assigned to the job. The composition of the identity string is shown in figure 16 and essentially consists of four comma separated substrings.

```
IdentityString='('$VO_NAME','$FQAN','$SUBJECT','$QUEUE')'
```

Figure 16: identity string to be matched

The XML file shown in figure 17 is changed to fit group name definitions in HTCondor's configuration. In the first line, it looks if the identity string contains /cms/dcms/ in its second substring and the result is group_cms.dcms, assuming cms is in the first substring (which it always should). If the identity string does not match the match expression, the next expression is checked. If this also does not match, it defaults to the first substring and appends it to group_.

5.5 High-Priority Jobs

In order to let high-priority jobs run as soon as possible, a slot is reserved for them. In HTCondor's default configuration there are as many slots as cores on the machine. To both enable jobs to use multiple cores and to not waste

```
<group match="^\(([^,]+),\S+\/cms\/dcms\/\S+,[^,]+,[^,]+\)$"
     result="group_$1.dcms"/>
<group match="^\(([^,]+),\S+\/cms\/\S+,[^,]+,[^,]+\)$"
    result="group_$1.cms"/>
<group match="^\(([^,]+)" result="group_$1"/>
```

Figure 17: matching identity string to groups

valuable CPU time, a dedicated slot configuration (see figure 18) for the execution machine is done. The configuration macro `NUM_CPUS` is changed to be one more than the actual number of cores. As the testing machine has 24 CPU cores, the macro `NUM_CPUS` is set to 25. This allows all 24 CPU cores to be utilized in normal conditions, and high-priority jobs borrow CPU time when executed. Slot 1 is assigned 24/25 of all resources (memory, disk, swap, and all cpu cores) and made to a partitionable slot, so jobs get as many resources as requested and get a slot that is forked off the main slot. The partitionable slot then advertises the remaining resources for further matchmaking. A second slot is assigned one (virtual) core and its slot-specific `START` expression is set to only accept high-priority jobs, i.e. jobs that come from pre-defined users or groups or have the ClassAd attribute `CreamQueue = "short"`. The partitionable slot's `START` expression is in turn set to not accept such jobs. Even though the groups `group_ops` and `group_dteam` are not defined in the central manager's configuration and therefore do not share an accounting group, their job ClassAds have their group defined and thus can be assigned to slot type 2.

```
NUM_CPUS = 24 + 1
NUM_SLOTS = 2

NUM_SLOTS_TYPE_1 = 1
SLOT_TYPE_1 = 24/25
SLOT_TYPE_1_PARTITIONABLE = true
SLOT_TYPE_1_START = $(START) && \
  (stringListIMember(TARGET.Owner, \
  "cmss cmsp014 cmsp015 cmspi043 cmspi058") \
  || (TARGET.AcctGroup =?= "group_ops") \
  || (TARGET.AcctGroup =?= "group_dteam")\
  || (TARGET.CreamQueue =?= "short")) =!= true

NUM_SLOTS_TYPE_2 = 1
SLOT_TYPE_2 = cpu=1, 1/25
SLOT_TYPE_2_PARTITIONABLE = false
SLOT_TYPE_2_START = $(START) && \
  (stringListIMember(TARGET.Owner, \
  "cmss cmsp014 cmsp015 cmspi043 cmspi058") \
  || (TARGET.AcctGroup =?= "group_ops") \
  || (TARGET.AcctGroup =?= "group_dteam")\
  || (TARGET.CreamQueue =?= "short"))
```

Figure 18: slot configuration on an execution machine with 24 CPU cores

5.6 Limiting Job Runtime

Jobs that need too long to finish need to be terminated. Their time limit is set to three days real time. Running jobs keep running until both `SUSPEND` and `WANT_SUSPEND` are `true`, or if `PREEMPT` is `true`, or the jobs terminate. Setting the `PREEMPT` macro to be `true` three days after job start will force jobs to free the slot (see figure 19). `SUSPEND` and `WANT_SUSPEND` are set to false, as these are settings designed for use on working stations, where the machine's owner returns to the computer and needs the computing resources for personal use. Preempted jobs may be vacated (soft-killed), allowing the job's processes to clean up and terminate itself. This is disabled however, as typical jobs sent to the grid cluster are not designed to handle soft-killing, and should not run for nearly three days. The macro `WANT_VACATE` is therefore set to `false`, leading to a hard-kill of all processes belonging to the job.

```
MINUTE           = 60
HOUR             = (60 * $(MINUTE))
DAY              = (24 * $(HOUR))
ActivationTimer  = ifThenElse(JobStart =!= UNDEFINED, (time
    () - JobStart), 0)
START = true
SUSPEND = false
PREEMPT = $(ActivationTimer) > 3*$(DAY)
WANT_SUSPEND = false
WANT_VACATE = false
```

Figure 19: preempting and killing jobs that last for three days

5.7 Priority Decay

In order to let the priority decay at the same rate as was the case with Maui, the `PRIORITY_HALFLIFE` had to be changed. It defaults to one day (86400 seconds). The priority decay from Maui's configuration had to be recalculated into a halftime. The resulting 1136819 seconds is then set as HTCondor's `PRIORITY_HALFLIFE`.

$$
\begin{aligned}
& 0.9^{\delta_t/(2\text{ days})} && = 0.5^{\delta_t/\text{halflife}} \\
\Rightarrow\ & \delta_t/(2\text{ days}) \cdot \ln(0.9) && = \delta_t/\text{halflife} \cdot \ln(0.5) \\
\Leftrightarrow\ & \text{halflife} && = 2\text{ days} \cdot \ln(0.5)/\ln(0.9) \\
& && = 13.157\text{ days} \\
& && = 1136819\text{ seconds}
\end{aligned}
$$

6 Testing

6.1 Setup

A pair of machines were made available to test HTCondor configuration settings. Machine `grid-tmp5` acts as the central manager, and machine `grid-wn048` acts as the execution machine. The grid manager set them up to mostly the same settings as on the productive grid system, with the exception of CREAM being set to submit jobs via HTCondor and using Quattor's configuration options for HTCondor and CREAM.

For testing purposes, the correct quotas (see figure 15 in section 5.3) are commented out and fictional values are set instead (see figure 20). Those values are multiples of 1/24, so their quotas represent an integer number fraction of 24 total available slots, or multiples of 1/18 for an integer number fraction of 18 slots assigned for `group_cms`. This change was necessary, as the negotiator will not assign jobs to slots if the assigned quota results to less than a slot in its first round of a negotiation cycle.

The groups `group_auger` and `group_icecube` had a quota of 0.0144, which would result to around a third of a slot.

```
GROUP_QUOTA_DYNAMIC_group_auger     = 3.0/24
GROUP_QUOTA_DYNAMIC_group_icecube   = 3.0/24
GROUP_QUOTA_DYNAMIC_group_cms       =18.0/24
GROUP_QUOTA_DYNAMIC_group_cms.dcms =12.0/18
GROUP_QUOTA_DYNAMIC_group_cms.cms   = 6.0/18
```

Figure 20: group quotas for testing

The configuration option `NEGOTIATOR_DEBUG = D_FULLDEBUG` was set to enable a detailed log file to see the negotiation process in great detail. This allowed to inspect the negotiation process to a detail not described in HTCondor's documentation.

6.2 Used Programs

In order to test the configuration for HTCondor, different programs were used. Some useful command line programs provided by HTCondor were used; these are `condor_submit` to submit jobs, `condor_status` to see the available slots and to get execution machine ClassAds, `condor_q` to see the job queue and to get job ClassAds, `condor_config_val` to see current and default values of configuration macros and especially configuration templates, `condor_reconfig` to reread the configuration file, `condor_restart` to restart single or all daemons, and `condor_userprio` to see the user priority values of individual users and their groups.

To submit jobs, submission files and programs had to be written. The program to be sent as jobs was a simple script, which calculated something for a short amount of time and suspended execution for ten minutes (shown in figure 21). The suspension time was later randomized, as the simultaneous termination of jobs was suspected to cause the negotiator to assign more slots to

a group than their quota suggests.

```
#! /usr/bin/python
from sys import argv
from time import sleep
from random import uniform

sleeptime = 500 + uniform(0, 200)
sleep(sleeptime)

result=1.0
x=float(argv[1])
inc=1.0
for n in range(1000):
        inc*=x/(n+1)
        result += inc
print result
```

Figure 21: python script (`exp.py`)

The submission file used for submitting was the same as in figure 10, but a small shell script (figure 22) was used to submit jobs as different users, adding the corresponding `accounting_group`, and submitting appropriate numbers of this job to the queue.

```
sudo -u cms001 condor_submit -append "accounting_group=
    group_cms.cms" submitfile -queue 2500
sudo -u dcms001 condor_submit -append "accounting_group=
    group_cms.dcms" submitfile -queue 5000
sudo -u aug001 condor_submit -append "accounting_group=
    group_auger" submitfile -queue 1250
sudo -u ice001 condor_submit -append "accounting_group=
    group_icecube" submitfile -queue 1250
sudo -u cmsp014 condor_submit -append "accounting_group=
    group_cms.cms" submitfile -queue 1
sudo -u ops001 condor_submit -append "accounting_group=
    group_ops" submitfile -queue 1
sudo -u dte001 condor_submit -append "accounting_group=
    group_dteam" submitfile -queue 1
sudo -u dcms001 condor_submit -append "accounting_group=
    group_cms.dcms" -append "+CreamQueue=\"short\""
    submitfile -queue 1
```

Figure 22: a small shell script to submit multiple jobs as different users

Another program used for testing and inspection was `condorview` [31]. It is an old contribution module to HTCondor and consists of shell scripts, HTML pages and a Java applet. When properly set up, `condorview` regularly queries HTCondor's current status and writes information about users and the amount of their idle and running jobs into data files. The Java applet built into the HTML pages reads the data files and graphically displays running or idle jobs

for up to eight users. This Java applet reads different data files based on the parameters given within the different HTML pages. Unfortunately, most current web browsers do not support Java applets anymore, so the applet is started using IcedTea-Web (a free software variant of Java Web Start)[32] while giving the corresponding HTML page as a command line argument.

6.3 Results

HTCondor improves memory usage and time on job submission and negotiation by gathering jobs into clusters. Unfortunately, this only happens when a submission file declares more than one job in `queue`, which CREAM does not do. Therefore, a simple shell script was written to simulate the submission of 10000 separate jobs, and time was measured for the submission of the 10001^{st} job. The resulting time of 0.3 seconds (see figure 23) is considerably less than a minute Maui and TORQUE needed for submitting a job with approximately 5000 jobs in the queue[7].

```
submitting 10000 jobs:
real    18m49.949s
user    8m22.545s
sys     5m10.333s
time used for submitting another job with 10000 jobs queued:
real    0m0.257s
user    0m0.048s
sys     0m0.187s
```

Figure 23: time needed to submit 10000 single jobs and a job after that

An unfortunate intermediate result was the group quota assignments. Group accounting itself worked, as groups were negotiated one after another, and priority took effect only in the order of users within a group. The Negotiator's log file shows this process. A problem arised with allowing surplus between groups. The Negotiator allocated double the amount of slots for groups `group_cms.cms` and `group_cms.dcms`, leading to the first negotiated group to get all allocated slots, if available. In this case, the dynamic quotas of the groups were calculated into corresponding number of slots—e.g. `group_cms.cms` got 6 slots and `group_cms.dcms` got 12 slots—and the number of allocated slots was doubled to 12 and 24 slots respectively. Groups `group_auger` and `group_icecube` were negotiated first, allocated the number of slots according to their quota, and were assigned to this amount of execution machines. Group `group_cms.cms` was negotiated next, got machines according to all 12 allocated slots, and `group_cms.dcms` was left with only 6 available slots.

Figure 24 shows the number of running jobs for each user in the first hour after mass submission. As the jobs from user `cms001` are the first to be submitted in the shell script, they are the first to be negotiated and assigned to slots on execution machines. Jobs from other groups follow soon thereafter. Whenever the total amount of running jobs exceeds 24, the specially configured slot is used to execute a high-priority job. The drops in the total number of running jobs are free slots due to finished jobs, where the negotiator has not finished assigning new jobs.

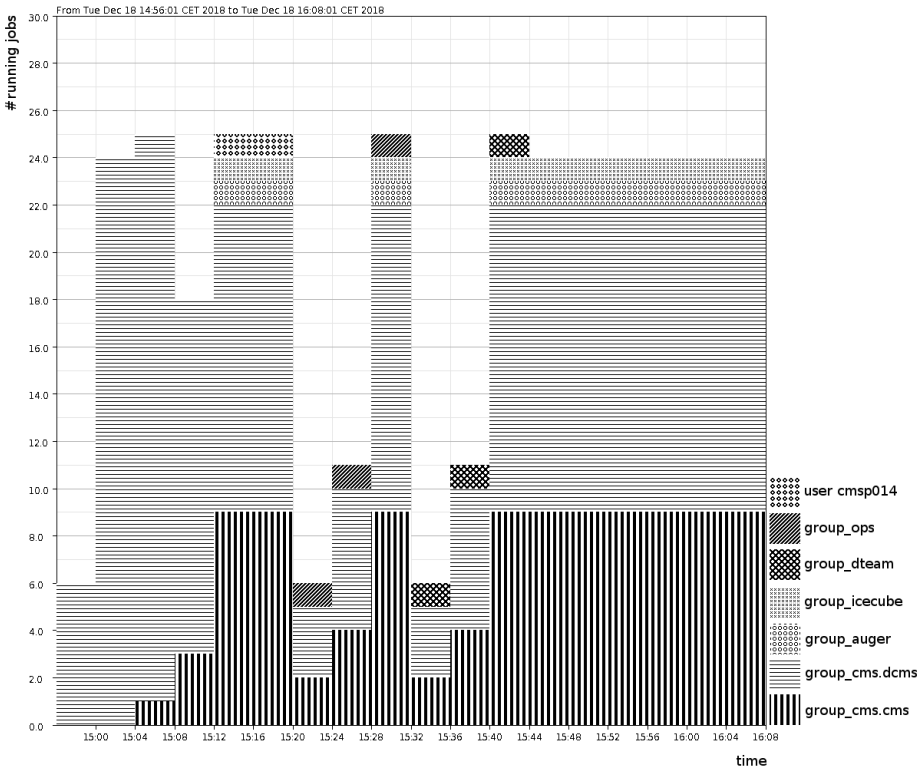

Figure 24: number of running jobs per group in the first hour after submitting

The symptom was caused by the jobs terminating at the same time and therefore freeing many execution slots at once. The submitted python script was then modified to suspend execution for a random amount of time between 500 seconds and 700 seconds. This leads to a better and more realistic slot assignment, as is shown in figure 25.

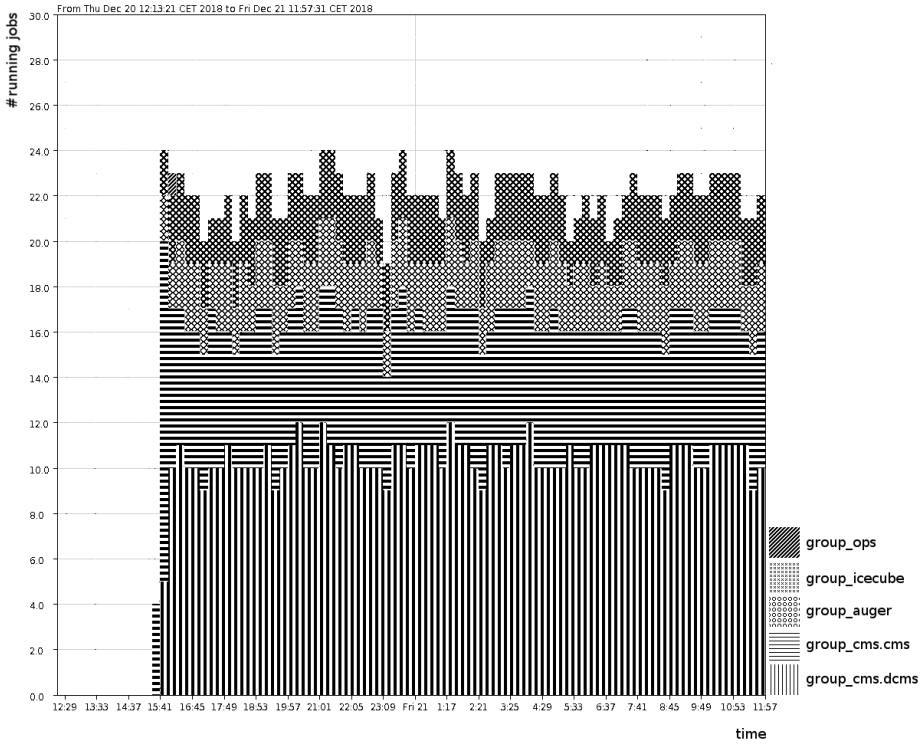

Figure 25: running jobs per group, after randomization of termination time

7 Comparing Previous with Current Setup

HTCondor has most of the functionality TORQUE and Maui provide, but sometimes needs different structures or algorithms to gain the same effect. The time limit of jobs is set to three days real time. HTCondor cannot set different time limitations for different groups or queues. Limiting CPU time might be possible for HTCondor, and this may be a topic for future improvements (see section 8). Instead of TORQUE's multiple queues and Maui's fair-share targets, HTCondor uses a single queue and divides submitters into different groups, assigning them quotas, thus achieving a similar result. Maui's fair-share decay is translated into HTCondor's simpler user priority half-life. HTCondor scales better with the number of jobs than TORQUE and Maui, allowing fast job submission even with one hundred thousand jobs in the queue. HTCondor is also actively developed and has newer software updates than TORQUE and especially Maui. A tabular overview is given in figure 26.

8 Future Improvements

Now that TORQUE and Maui are replaced with HTCondor and tested on two test machines, HTCondor should be installed on the main grid cluster and its configuration should be copied over with slight changes (Names, IP addresses) and carefully set as the job scheduler for this grid cluster, while still having the

	Maui & TORQUE	HTCondor
real time limit	73 hours & 4 hours	72 hours
CPU time limit	49 hours & 2h 10m	(none)
groups	three layers	hierarchical
quotas	working	working
priority decay	to 90% after two days	half-life of 13 days
max #jobs	5000	10,000+
job submission time	ca. 1 minute [7]	< 0.3 second
last software update	ca. 2014 (Maui) [27] and 21.3. 2018 (TORQUE) [28]	31.10. 2018 [16]

Figure 26: comparison overview

option to revert to the old setting with TORQUE and Maui.

The limitation of jobs to two days of CPU time could be achieved by using resource management through linux control groups (cgroups)[29]. HTCondor's configuration macro `CGROUP_MEMORY_LIMIT_POLICY`[30] enables memory limit policies to be enforced by cgroups. A similar approach could possibly be made by configuring cgroups to limit CPU time, but it needs a dedicated cgroups configuration.

A better and visually more pleasing job monitor could be added. The command line tools `condor_status` and `condor_q` only give information about the current state of the pool and queue, and `condorview` relies the old and outdated technology of Java applets. HTCondor has an optional daemon named `GANGLIA_D`,[33] which may be used in conjecture with the monitoring system Ganglia[34].

A Configuration Files

A.1 Central Manager

```
########################################
# settings already set by quattor in condor_config and
# the various quattor_*.conf files
########################################

UID_DOMAIN = physik.rwth-aachen.de
COLLECTOR_NAME = "Condor Cluster at $(UID_DOMAIN)"
CONDOR_ADMIN = root@$(FULL_HOSTNAME)
CONDOR_HOST = grid-tmp5.physik.rwth-aachen.de
RELEASE_DIR = /usr
LOCAL_DIR = /var
LOCAL_CONFIG_FILE = /etc/condor/condor_config.local
REQUIRE_LOCAL_CONFIG_FILE = False
LOCAL_CONFIG_DIR = /etc/condor/config.d
DAEMON_LIST = MASTER COLLECTOR SCHEDD



##  Pathnames
RUN     = $(LOCAL_DIR)/run/condor
LOG     = $(LOCAL_DIR)/log/condor
LOCK    = $(LOCAL_DIR)/lock/condor
SPOOL   = $(LOCAL_DIR)/lib/condor/spool
EXECUTE = $(LOCAL_DIR)/lib/condor/execute
BIN     = $(RELEASE_DIR)/bin
LIB = $(RELEASE_DIR)/lib64/condor
INCLUDE = $(RELEASE_DIR)/include/condor
SBIN    = $(RELEASE_DIR)/sbin
LIBEXEC = $(RELEASE_DIR)/libexec/condor
SHARE   = $(RELEASE_DIR)/share/condor
PROCD_ADDRESS = $(RUN)/procd_pipe
JAVA_CLASSPATH_DEFAULT = $(SHARE) $(SHARE)/scimark2lib.jar .
SSH_TO_JOB_SSHD_CONFIG_TEMPLATE = /etc/condor/
    condor_ssh_to_job_sshd_config_template
```

```
use SECURITY : HOST_BASED
ALLOW_WRITE = *.physik.rwth-aachen.de
# This is to enforce password authentication
SEC_DAEMON_AUTHENTICATION = required
SEC_DAEMON_AUTHENTICATION_METHODS = password
SEC_CLIENT_AUTHENTICATION_METHODS = password,fs,gsi
SEC_PASSWORD_FILE = /var/lib/condor/condor_credential
ALLOW_DAEMON = condor_pool@*
TRUST_UID_DOMAIN = TRUE



MAXMEM = 2000
MAXWALLTIME = 4320
SERVINGSTATE = Production
NEGOTIATOR_ENABLE=true
if $(NEGOTIATOR_ENABLE)
    DAEMON_LIST = $(DAEMON_LIST) NEGOTIATOR
endif



NEGOTIATOR_POST_JOB_RANK = \
   (RemoteOwner =?= UNDEFINED) * \
   (ifThenElse(isUndefined(Mips), 1000, Mips) - \
   SlotID - 1.0e10*(Offline=?=True))

ROTATE_HISTORY_DAILY = true
MAX_HISTORY_ROTATIONS = 1000

SYSTEM_PERIODIC_REMOVE = (JobStatus == 5 && time() -
    EnteredCurrentStatus > 3600*48)\
|| ((JobStatus == 2)&&(($(MAXWALLTIME)>0)&&((time() -
    EnteredCurrentStatus) > (60*$(MAXWALLTIME)))))\



########################################
# security settings
########################################
# Enter "condor_config_val use SECURITY:HOST_BASED"
# to see what macros/values this templates sets
use SECURITY : HOST_BASED
ALLOW_READ   = grid-*.physik.rwth-aachen.de
ALLOW_DAEMON = condor_pool@physik.rwth-aachen.de



# testwise disable daemon authentication
#SEC_DAEMON_AUTHENTICATION = optional
#SEC_CLIENT_AUTHENTICATION = optional
```

```
##########################################
# some definitions that come handy later on
##########################################
MINUTE          = 60
HOUR            = (60 * $(MINUTE))
STANDARD        = 1
VANILLA         = 5



##########################################
# collector config attributes
##########################################
# enable writing statistics
KEEP_POOL_HISTORY           = True
POOL_HISTORY_DIR            = /var/log/condor/log_statistics
COLLECTOR_DAEMON_STATS      = True
COLLECTOR_QUERY_WORKERS     = 16



##########################################
# negotiator config attributes
##########################################
# enable verbose output in negotiator logfile
# /var/log/condor/NegotiatorLog
NEGOTIATOR_DEBUG = D_FULLDEBUG
PRIORITY_HALFLIFE = 1136819



##########################################
# group accounting
# groups originally taken from /etc/condor/groups_list
# group matching done in /usr/libexec/
    condor_local_submit_attributes.sh
# and changed in /etc/condor/groups_mapping.xml
##########################################
GROUP_NAMES = group_cms, group_cms.cms, group_cms.dcms, \
              group_auger, group_icecube
GROUP_ACCEPT_SURPLUS = true
GROUP_AUTOREGROUP = false



#GROUP_QUOTA_DYNAMIC_group_auger    = 0.014408562902582258
#GROUP_QUOTA_DYNAMIC_group_cms      = 0.9711828741948355
#GROUP_QUOTA_DYNAMIC_group_cms.cms  = 0.347098775649
#GROUP_QUOTA_DYNAMIC_group_cms.dcms = 0.652901224351
#GROUP_QUOTA_DYNAMIC_group_icecube  = 0.014408562902582258
#GROUP_QUOTA_DYNAMIC_group_dteam    = 0.0001
#GROUP_QUOTA_DYNAMIC_group_ops      = 0.0001
```

```
GROUP_QUOTA_DYNAMIC_group_auger    = 3.0/24
GROUP_QUOTA_DYNAMIC_group_icecube  = 3.0/24
GROUP_QUOTA_DYNAMIC_group_cms      =18.0/24
GROUP_QUOTA_DYNAMIC_group_cms.dcms =12.0/18
GROUP_QUOTA_DYNAMIC_group_cms.cms  = 6.0/18
```

A.2 Execution Machines

```
##########################################
# security settings
##########################################
use SECURITY : HOST_BASED
ALLOW_WRITE = $(ALLOW_WRITE) condor@$(CONDOR_HOST) root@$(
    CONDOR_HOST)
ALLOW_READ  = grid-*.physik.rwth-aachen.de



##########################################
# some definitions that come handy later on
##########################################
MINUTE           = 60
HOUR             = (60 * $(MINUTE))
DAY              = (24 * $(HOUR))
STANDARD         = 1
VANILLA          = 5
BackgroundLoad   = 0.3
HighLoad         = 0.5
StartIdleTime    = 5 * $(MINUTE)
ContinueIdleTime =  5 * $(MINUTE)
MaxSuspendTime   = 12 * $(HOUR)
MaxVacateTime    = 10 * $(MINUTE)
StateTimer       = (time() - EnteredCurrentState)
ActivityTimer    = (time() - EnteredCurrentActivity)
ActivationTimer  = ifThenElse(JobStart =!= UNDEFINED, (time
    () - JobStart), 0)
```

```
########################################
# startd config attributes
########################################
START = true
SUSPEND = false
PREEMPT = $(ActivationTimer) > 3*$(DAY)
WANT_HOLD = false
WANT_SUSPEND = false
WANT_VACATE = false
KILL = true
POLLING_INTERVAL = $(MINUTE)/2
CLAIM_WORKLIFE = 0



########################################
# NUM_CPUS shall be set by quattor, because SLOT_TYPE_<N>
# needs to be typed as [resource]=abs_value, or as
    percentage%,
# or as a fraction numerator/denominator, but 1/$(NUM_CPUS)
# is not accepted. Additionally, SLOT_TYPE_1
# may not be undefined and cannot be changed during
# condor_startd runtime (not even through condor_reconfig;
# use condor_restart -startd instead)
########################################

# machine specific
NUM_CPUS = 24 + 1
#NUM_CPUS = $(DETECTED_CPUS) + 1
NUM_SLOTS=2



NUM_SLOTS_TYPE_1 = 1
SLOT_TYPE_1 = 24/25
#SLOT_TYPE_1 = ($(NUM_CPUS)-1.0)/$(NUM_CPUS)
SLOT_TYPE_1_PARTITIONABLE = true
SLOT_TYPE_1_START = $(START) && (stringListIMember(TARGET.
    Owner, "cmss cmsp014 cmsp015 cmspi043 cmspi058") \
  || (TARGET.AcctGroup =?= "group_ops") || (TARGET.AcctGroup
      =?= "group_dteam")\
  || (TARGET.CreamQueue =?= "short")) =!= true



NUM_SLOTS_TYPE_2 = 1
SLOT_TYPE_2 = cpu=1, 1/25
#SLOT_TYPE_2 = cpu=1, 1.0/$(NUM_CPUS)
SLOT_TYPE_2_PARTITIONABLE = false
SLOT_TYPE_2_START = $(START) && (stringListIMember(TARGET.
    Owner, "cmss cmsp014 cmsp015 cmspi043 cmspi058") \
  || (TARGET.AcctGroup =?= "group_ops") || (TARGET.AcctGroup
      =?= "group_dteam")\
  || (TARGET.CreamQueue =?= "short"))
```

References

[1] Worldwide LHC Computing Grid
http://wlcg-public.web.cern.ch/about

[2] WLCG Tiers
http://wlcg-public.web.cern.ch/tier-centres

[3] Computing Resource Execution And Management (CREAM)
https://twiki.cern.ch/twiki/bin/view/EGEE/GLiteCREAMCE

[4] Unified Middleware Distribution
http://repository.egi.eu/

[5] TORQUE
https://www.adaptivecomputing.com/products/torque/

[6] Maui
https://www.adaptivecomputing.com/products/maui/

[7] personal communication with: Dr. Andreas Nowack,
Grid Manager RWTH Aachen University, Physics Institute III.B

[8] History of Portable Batch System
https://www.pbsworks.com/AboutGT.aspx?d=About,-History

[9] Altair PBS Professional (commercial version)
https://www.pbsworks.com/PBSProduct.aspx?n=
Altair-PBS-Professional&c=Overview-and-Capabilities

[10] PBS Professional Open Source Project
https://www.pbspro.org/

[11] Software Patches for OpenPBS
https://www.mcs.anl.gov/research/projects/openpbs/

[12] IBM Spectrum LSF
https://www.ibm.com/support/knowledgecenter/en/SSWRJV/
product_welcome_spectrum_lsf.html

[13] Grid Engine Wiki
http://wiki.gridengine.info/wiki/

[14] Slurm
https://slurm.schedmd.com/quickstart.html

[15] Maui's fair-share settings
http://docs.adaptivecomputing.com/maui/6.3fairshare.php

[16] web page for HTCondor
http://research.cs.wisc.edu/htcondor/

[17] Description of HTCondor
http://research.cs.wisc.edu/htcondor/description.html

[18] List and short description of all HTCondor daemons
http://research.cs.wisc.edu/htcondor/manual/v8.6/3_1Introduction.html

[19] Description of ClassAds
http://research.cs.wisc.edu/htcondor/manual/v8.6/4_1HTCondor_s_ClassAd.html

[20] List of nearly all configuration macros
http://research.cs.wisc.edu/htcondor/manual/v8.6/3_5Configuration_Macros.html

[21] System administration toolkit Quattor
https://www.quattor.org/

[22] Quattor's configuration templates for HTCondor
https://github.com/quattor/template-library-grid/tree/umd-4/features/htcondor/templ

[23] HTCondor's security settings
http://research.cs.wisc.edu/htcondor/manual/v8.6/3_8Security.html

[24] User priority and negotiation
http://research.cs.wisc.edu/htcondor/manual/v8.6/3_6User_Priorities.html

[25] How to configure priorities/quotas for groups of users [using machine ranks in HTCondor]
http://htcondor-wiki.cs.wisc.edu/index.cgi/wiki?p=HowToConfigPrioritiesForUsers

[26] How to configure a user's fair share of the pool
http://htcondor-wiki.cs.wisc.edu/index.cgi/wiki?p=HowToConfigureFairShare

[27] Maui Administrator's Guide
http://docs.adaptivecomputing.com/maui/index.php

[28] TORQUE versions and release dates
http://www.adaptivecomputing.com/support/documentation-index/torque-resource-manager-documentation/

[29] Resource management guide for using cgroups (document by RedHat Enterprise Linux)
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/

[30] Cgroup-Based Process Tracking in HTCondor
http://research.cs.wisc.edu/htcondor/manual/v8.6/3_14Setting_Up.html#SECTION00414120000000000000

[31] The HTCondorView Client Contrib Module
http://research.cs.wisc.edu/htcondor/manual/v8.6/9_4HTCondorView_Client.html

[32] IcedTea-Web web page
https://icedtea.classpath.org/wiki/IcedTea-Web

[33] Using HTCondor's GANGLIA_D daemon for easier resource monitoring
http://research.cs.wisc.edu/htcondor/manual/v8.6/3_12Monitoring.html

[34] Ganglia
http://ganglia.info/