Introduction to HTCondor

How to distribute your compute tasks and get results with high performance, keeping machines and site admins joyful

Oliver Freyermuth

University of Bonn
freyermuth@physik.uni-bonn.de

28th August, 2019

Overview

1. Introduction
2. How HTCondor works and how it can be used
3. What might go wrong. . .
4. Hands-on tutorial!

Find this talk and the actual tutorial at:
`https://git.io/gridka-2019-htcondor`

Welcome!

About me

- studied physics in Bonn, starting in 2007
- PhD finished in 2017 at the BGO-OD experiment located at ELSA in Bonn (Hadron Physics, photoproduction)
  Focus on software development (C++ / ROOT)
- since 2017: IT dep. of Physikalisches Institut at Uni Bonn
  - Central services (desktops, printers, web, virtualization. . .)
  - **Grid-enabled computing cluster:**
    used by HEP, theory, detector dev., photonics,. . .
    HTCondor & Singularity containers, CephFS, CVMFS,. . .
  - Automation of all services and machine deployments
  - Support for users
  - IT security

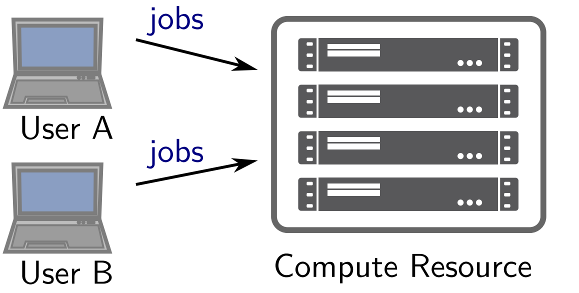
TL;DR: Feel free to ask both from user and admin point of view!

Now: Your turn!

HTCondor

- Workload Management system for dedicated resources, idle desktops, cloud resources, . . .
- Project exists since 1988 (named Condor until 2012)
- Open Source, developed at UW-Madison, Center for High Throughput Computing
- Key concepts:
    - '**Submit Locally. Run globally.**' (Miron Livny)
      One interface to any available resource.
    - Integrated mechanisms for **file transfer** to / from the job
    - '**Class Ads**', for submitters, jobs, resources, daemons, . . .
      Extensible lists of attributes (expressions) — more later!
    - Supports Linux, Windows and MacOS X and has a very diverse user base
      CERN community, Dreamworks and Disney, NASA,. . .

What is a workload manager?



(e.g. local cluster, desktops, cloud)

- takes care of collecting user's requirements
- prioritization / fair share
- enforcing limits
- collect resource information
- distribute jobs efficiently
- monitor status for users and admins

Why HTCondor?

High Throughput Computing

many jobs, usually loosely coupled or independent, goal is large throughput of jobs and / or data

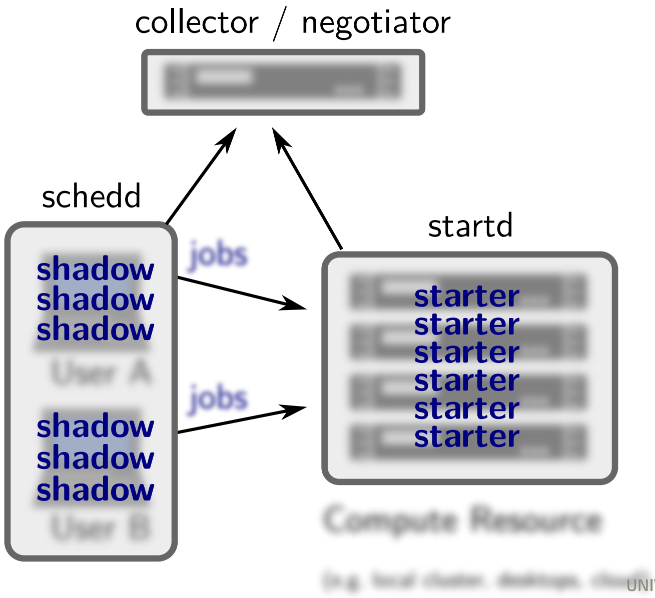
High Performance Computing

tightly coupled parallel jobs which may span several nodes and often need low-latency interconnects

- HTCondor can do both *(HPC-like tasks need some 'tuning')*
- HPC community: *Slurm* (less flexible, but easier to get up and running for HPC!)

⇒ Let's have a look at how HTCondor works.

Structure of HTCondor



HTCondor's ClassAds

- Any submitter, job, resource, daemon has a ClassAd
- ClassAds are basically just expressions (`key = value`)
- Dynamic evaluation and merging possible

Job ClassAd

```
Executable = some-script.sh
+ContainerOS="CentOS7"

Request_cpus = 2
Request_memory = 2 GB
Request_disk = 100 MB
```

Machine ClassAd

```
Activity = "Idle"
Arch = "X86_64"
Cpus = 8
DetectedMemory = 7820
Disk = 35773376
has_avx = true
has_sse4_1 = true
has_sse4_2 = true
has_ssse3 = true
KFlops = 1225161
Name = "slot1@htcondor-wn-7"
OpSys = "LINUX"
OpSysAndVer = "CentOS7"
OpSysLegacy = "LINUX"
Start = true
State = "Unclaimed"
```

UNIVERSITÄT BONN

HTCondor's ClassAds

- Job and Machine ClassAd extended / modified by HTCondor configuration
- Merging these ClassAds determines if job can run on machine
- Examples for dynamic parameters:
  - Select a different binary depending on OS / architecture
  - Machine may only want to 'Start' jobs from some users
- You can always check out the ClassAds manually to extract all information (use the argument `-long` to commands!)
- To extract specific information, you can tabulate specific attributes:

```
$ condor_q -all -global -af:hj Cmd ResidentSetSize_RAW
↪  RequestMemory RequestCPUs
 ID      Cmd         ResidentSetSize_RAW RequestMemory RequestCPUs
   2.0   /bin/sleep 91168               2048          1
```

What HTCondor needs from you...

A job description / Job ClassAd

Resource request, environment, executable, number of jobs,...

```
Executable = some-script.sh

Arguments  = some Arguments for our program $(ClusterId) $(Process)
Universe   = vanilla
Transfer_executable     = True

Error                   = logs/err.$(ClusterId).$(Process)
#Input                  = input/in.$(ClusterId).$(Process)
Output                  = logs/out.$(ClusterId).$(Process)
Log                     = logs/log.$(ClusterId).$(Process)

+ContainerOS="CentOS7"

Request_cpus = 2
Request_memory = 2 GB
Request_disk = 100 MB

Queue
```

What HTCondor needs from you. . .

some-script.sh

- Often, you want to use a wrapper around complex software
- This wrapper could be a shell script, python script etc.
- It should take care of:
  - Argument handling
  - Environment setup (if needed)
  - Exit status check (bash: consider -e)
  - Data handling (e.g. move output to shared file system)

```
#!/bin/bash
source /etc/profile
set -e
SCENE=$1

cd ${SCENE}
povray +V render.ini
mv ${SCENE}.png ..
```

Submitting a job

```
$ condor_submit myjob.jdl
Submitting job(s)..
1 job(s) submitted to cluster 42.
```

There are many ways to check on the status of your job (we will try them in the tutorial):

- `condor_tail -f` can follow along `stdout` / `stderr` (or any other file in the job sandbox
- `condor_q` can access job status information (memory usage, CPU time,. . .)
- `log` file contains updates about resource usage, exit status etc.
- `condor_history` provides information after the job is done
- `condor_ssh_to_job` may allow to connect to the running job

Advanced JDL syntax

```
Executable = /home/olifre/advanced/analysis.sh
Arguments = "-i '$(file)'"
Universe = vanilla
if $(Debugging)
  slice = [:1]
  Arguments = "$(Arguments) -v"
endif
Error  = log/$Fn(file).stderr
Input  = $(file)
Output = log/$Fn(file).stdout
Log    = log/analysis.log
Queue FILE matching files $(slice) input/*.root
```

HTCondor also offers macros and can queue based on files.
Can you guess what happens if you submit like this?

```
condor_submit 'Debugging=true' analysis.jdl
```

DAGs: Directed Acyclic Graphs

- Often, jobs of different type of an analysis chain depend on each other
  Example: Monte Carlo, comparison to real data, Histogram merging,. . .
- These dependencies can be described with a DAG
- Condor runs a special ‘DAGMAN’ job which takes care of submitting jobs for each ‘node’ of the DAG,
  check status, limit idle and running jobs, report status etc.
  (like a *Babysitter job*)
- DAGMAN comes with separate logfiles, DAGs can be stopped and resumed

We will see an example in the tutorial!

Problems and inefficiencies

- Theoretically, users should not need to care about cluster details. . .
- Jobs *could* transfer all their data with them, and back — but this does not scale for GB of data, thousands of files for thousands of jobs
- Jobs need to take care to be ‘mobile’ and run in the correct environment

Some setup details can not be ignored for efficient usage

Let’s have a short look at elements of computing clusters and how (not) to design your jobs!

A typical HTC cluster: I/O intensive loads

- Shared / parallel file system for data, job input and output
  CephFS, Lustre, BeeGFS, GPFS,...
- Often, also a second file system (e.g. to distribute software)
  CVMFS, NFS, ...
- Usually, local scratch disks in all worker nodes
  'classic' file system such as ext4
- Often, dedicated submit nodes, data transfer nodes etc.

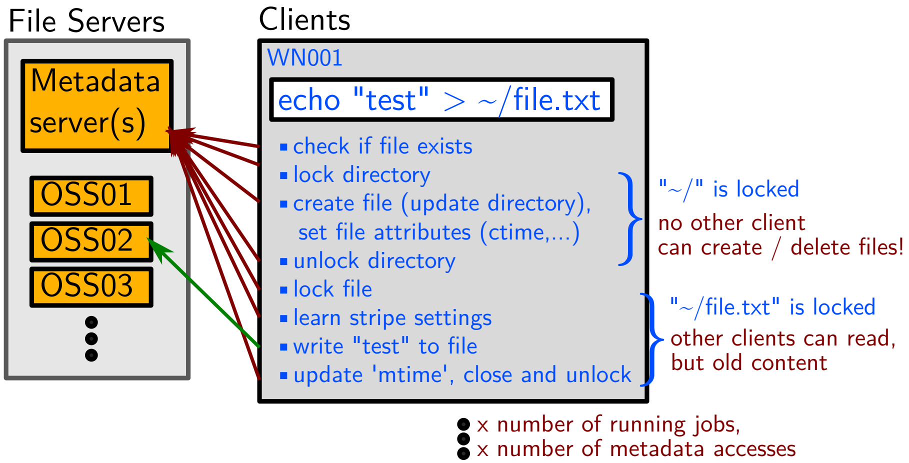
⇒ Lots of differently behaving file systems!

Working with a shared file system

Common sources of woes

- **Excessive file metadata operations**
  Syscalls: `open`, `close`, `stat`, `fsync`...)
  use `strace` *to diagnose and debug*
- **Storing or reading many small files from shared FS**
  There is usually a dedicated place for software (more later).
- **Destructive interference between jobs**
  - Opening an input file exclusively
  - Writing to the very same output file

Working with a shared file system


File Servers
Metadata server(s)
OSS01
OSS02
OSS03
Clients
WN001
echo "test" > ~/file.txt
check if file exists
lock directory
create file (update directory), set file attributes (ctime,...)
unlock directory
lock file
learn stripe settings
write "test" to file
update 'mtime', close and unlock
"~/" is locked
no other client can create / delete files!
"~/file.txt" is locked
other clients can read, but old content
x number of running jobs,
x number of metadata accesses

Working with a shared file system

Common solutions

- Use a different file system for software (many small files!)
  CVMFS, NFS,...
- Most software is (likely) already provided by cluster admins — use it!
  They know how to compile best for the available hardware.
- Do not install everything from scratch
  (e.g. `pip install "everything"`)
- Package quickly changing software builds in a tarball, extract it to scratch disk in the job wrapper script
  Advantage: Consistent software state for all jobs.
- Have jobs write to scratch first and move to shared FS later
  Advantage: If job is evicted, no broken output file.

Working with a workload manager

Common sources of woes

- Mismatched resource request and usage (more later)
- Hefty / bad use of condor file transfer, for example:
  Shared FS accessible from submit machine, transferring files from / to there
- Badly suited job runtimes

too short	Overhead per job causes inefficiency, some workload managers overload easily
too long	Unless the job does *checkpointing*, very sensitive to any disturbance, operational issues (kernel updates / reboots etc.)

- Frameworks which create thousands of JDL files and wrapper scripts
  (instead of using flexible syntax or Python API)

Working with different environments

How to compile code?

- Some resources may only be available via interactive jobs
  - Advantage for admins: No separate bare metal machines
  - Advantage for you: Environment the same as in the job!
- Compile the code, pack it into a tarball, send to shared FS / condor file transfer
- Can be automated with scripts / if offered, job start hooks (like '`.bashrc`')

Advantages of this approach

- Portable and stable job executables
- If combined with containers and 'mobile data': Mostly cluster independent jobs possible

Mismatched resource requests

Mismatched CPU request

- Often caused by software using all ‘visible’ cores — configure!

```
export NUMEXPR_NUM_THREADS=1
export MKL_NUM_THREADS=1
export OMP_NUM_THREADS=1
```

- Admins may export these variables for you. . .
- Too many threads: Congestion, may affect other jobs

Mismatched memory request

- Depending on configuration, may lead to swapping ⇒ hefty slowdown (affects also other jobs)
- Swap usage not visible in HTCondor Ads (yet)
- Admins could also set a hard limit (no swap) ⇒ job killed

What about other resources?

Disk Space

- Disk space is not 'consumable' in HTCondor
- Usually, this affects scratch space only (job working directory)
- Commonly, not an actual issue (shared file systems have quotas)
- More common is disk overload due to heavy syscalls / many small files / swap

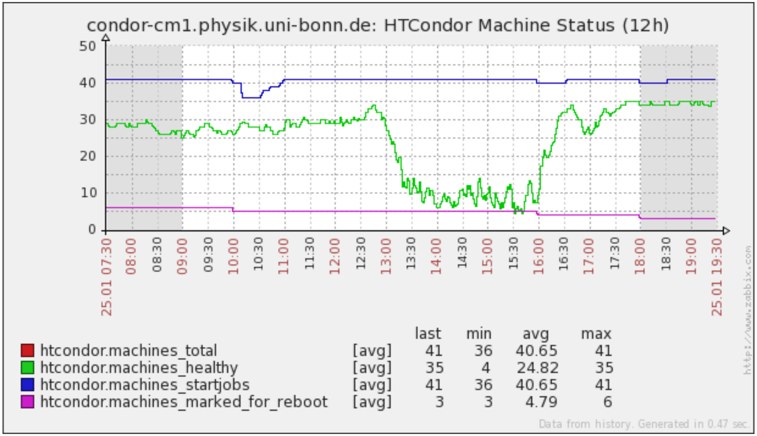
CPU cache thrashing

Commonly ignored issue — limiting e.g. CPU cache usage not supported by HTCondor yet (but there are plans)!

Common tricks used by admins

Node health check

Detects unhealthy node from error or misbehaving jobs.

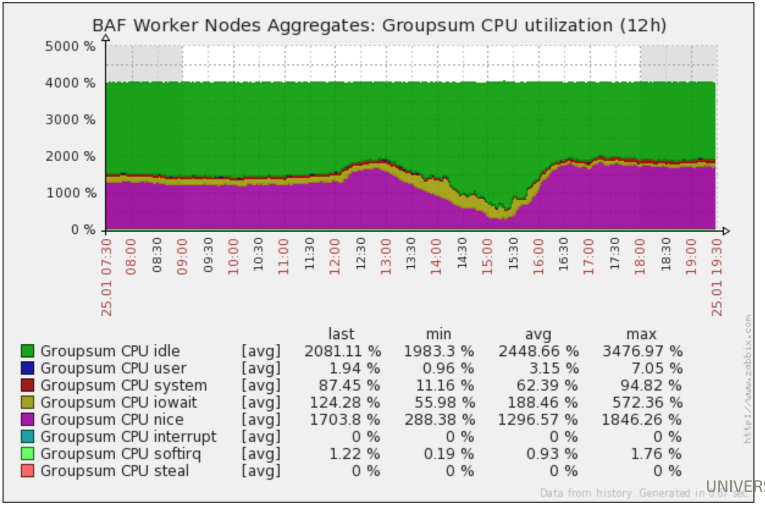


		last	min	avg	max
🟥 htcondor.machines_total	[avg]	41	36	40.65	41
🟩 htcondor.machines_healthy	[avg]	35	4	24.82	35
🟦 htcondor.machines_startjobs	[avg]	41	36	40.65	41
🟪 htcondor.machines_marked_for_reboot	[avg]	3	3	4.79	6

Common tricks used by admins

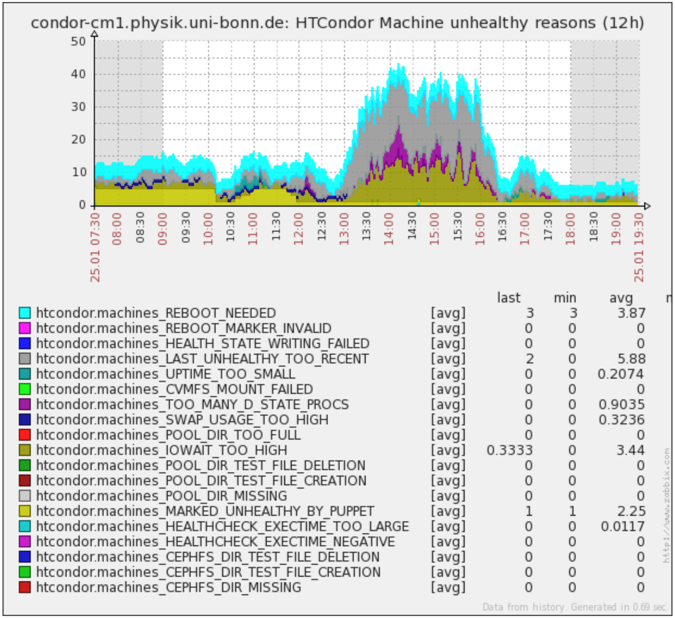
Node health check

Fights against spread of inefficiencies / overload.



		last	min	avg	max
🟩 Groupsum CPU idle	[avg]	2081.11 %	1983.3 %	2448.66 %	3476.97 %
🟦 Groupsum CPU user	[avg]	1.94 %	0.96 %	3.15 %	7.05 %
🟥 Groupsum CPU system	[avg]	87.45 %	11.16 %	62.39 %	94.82 %
🟨 Groupsum CPU iowait	[avg]	124.28 %	55.98 %	188.46 %	572.36 %
🟪 Groupsum CPU nice	[avg]	1703.8 %	288.38 %	1296.57 %	1846.26 %
🟦 Groupsum CPU interrupt	[avg]	0 %	0 %	0 %	0 %
🟩 Groupsum CPU softirq	[avg]	1.22 %	0.19 %	0.93 %	1.76 %
🟥 Groupsum CPU steal	[avg]	0 %	0 %	0 %	0 %

Common tricks used by admins



			last	min	avg
🟦	htcondor.machines_REBOOT_NEEDED	[avg]	3	3	3.87
🟪	htcondor.machines_REBOOT_MARKER_INVALID	[avg]	0	0	0
🟦	htcondor.machines_HEALTH_STATE_WRITING_FAILED	[avg]	0	0	0
⬜	htcondor.machines_LAST_UNHEALTHY_TOO_RECENT	[avg]	2	0	5.88
🟦	htcondor.machines_UPTIME_TOO_SMALL	[avg]	0	0	0.2074
🟩	htcondor.machines_CVMFS_MOUNT_FAILED	[avg]	0	0	0
🟪	htcondor.machines_TOO_MANY_D_STATE_PROCS	[avg]	0	0	0.9035
🟦	htcondor.machines_SWAP_USAGE_TOO_HIGH	[avg]	0	0	0.3236
🟥	htcondor.machines_POOL_DIR_TOO_FULL	[avg]	0	0	0
🟨	htcondor.machines_IOWAIT_TOO_HIGH	[avg]	0.3333	0	3.44
🟩	htcondor.machines_POOL_DIR_TEST_FILE_DELETION	[avg]	0	0	0
🟥	htcondor.machines_POOL_DIR_TEST_FILE_CREATION	[avg]	0	0	0
⬜	htcondor.machines_POOL_DIR_MISSING	[avg]	0	0	0
🟨	htcondor.machines_MARKED_UNHEALTHY_BY_PUPPET	[avg]	1	1	2.25
🟦	htcondor.machines_HEALTHCHECK_EXECTIME_TOO_LARGE	[avg]	0	0	0.0117
🟪	htcondor.machines_HEALTHCHECK_EXECTIME_NEGATIVE	[avg]	0	0	0
🟦	htcondor.machines_CEPHFS_DIR_TEST_FILE_DELETION	[avg]	0	0	0
🟩	htcondor.machines_CEPHFS_DIR_TEST_FILE_CREATION	[avg]	0	0	0
🟥	htcondor.machines_CEPHFS_DIR_MISSING	[avg]	0	0	0

Data from history. Generated in 0.69 sec.

UNIVERSITÄT BONN

Conclusion

- HTCondor is very flexible — you can check out configuration via ClassAds!
- Each cluster may be slightly different (CERN has job flavours to define job runtime, Bonn has containers with different environments,. . .)
- We will learn job submission today — to run efficiently, you also need to know your software and the cluster

Ask questions any time!

And now, get started at:
`https://git.io/gridka-2019-htcondor`

Thank you

for your attention!