'Startd flocking' (© ToddT) and 'HT-HPC' c/o the Milan Physics Dept.

Francesco Prelz, David Rebatto
INFN, sezione di Milano

Summary

- The AMICO Cluster at the Milan Physics Dept.
- Parallel workloads: what to do?
- A Unicorn for Christmas: parallel scheduling of Docker containers.
  - Can it - should it - does it work?
- On the pursuit of happiness.

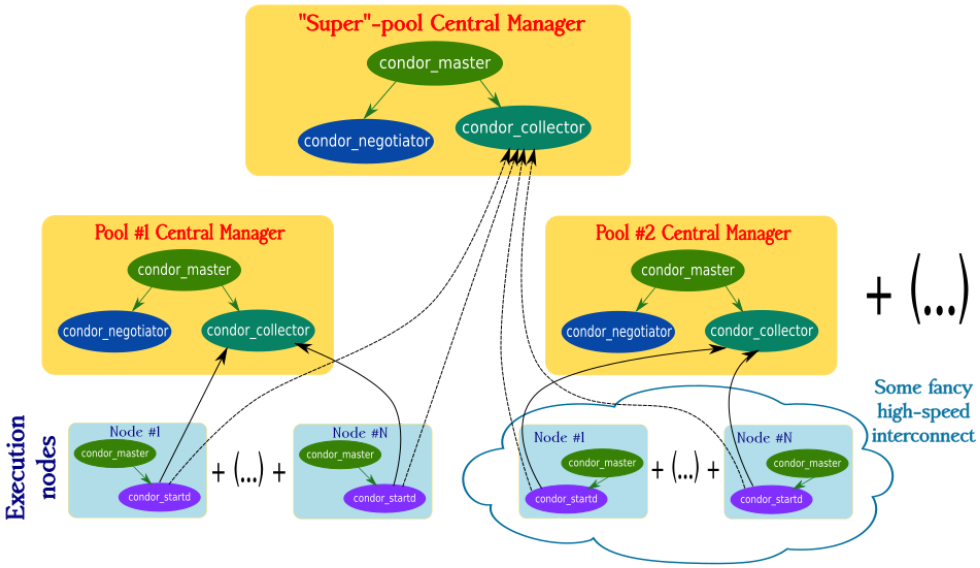
A snapshot of a Physics Dept in the fashion city.

- The Physics Department at the state university of Milan is a fairly large, 80-some faculty, 1150 student department. Its research activity is structured in a dozen 'groups', active in various fields (high-energy and nuclear Physics, solid state and condensed matter physics, astrophysics, theory, electronics, environmental and medical physics, etc...).
- Many of these groups proceeded in sparse order with the purchase of computing resources. The thought of rationalising this process came along quite *a posteriori*, but this should a be quite familiar scenario for people in this audience ...
- The typical purchase (with a couple notable exceptions - LHC Tier-2 centre, Theory group), would be a turn-key configuration of one rackful of worker nodes with an Infiniband interconnect, for the execution of MPI jobs.
- LHC Tier-2 cores: ~2500. Cores in the other 8 'group' clusters that were willing to cooperate into a common infrastructure, as of today: 1968.
- The Tier-2 center has been running Condor 'as a batch system' since the early days of WLCG (2002 or so): not investing into *other* technology sounded wise enough...

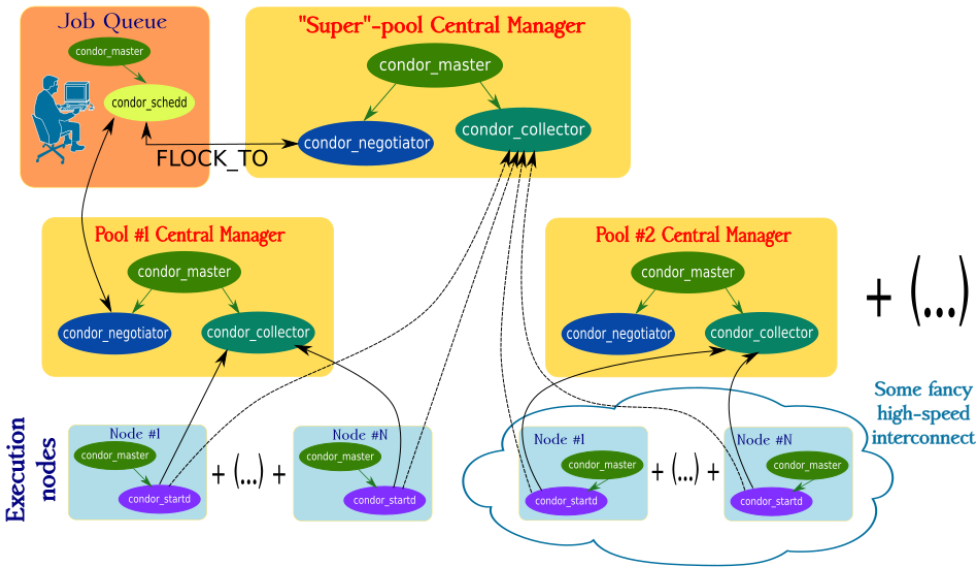
Enters: startd flocking

- Actually, we *did* look up the Condor Wiki, and were inspired by this entry: `https://htcondor-wiki.cs.wisc.edu/index.cgi/wiki?p=HowToHaveExecuteMachines`
- The description of what this solution is meant for is strikingly similar to the department scenario we just described...
- → Just throw into the picture partitionable slots - and possibly the submission of parallel jobs via the Dedicated Scheduler - more on this later.

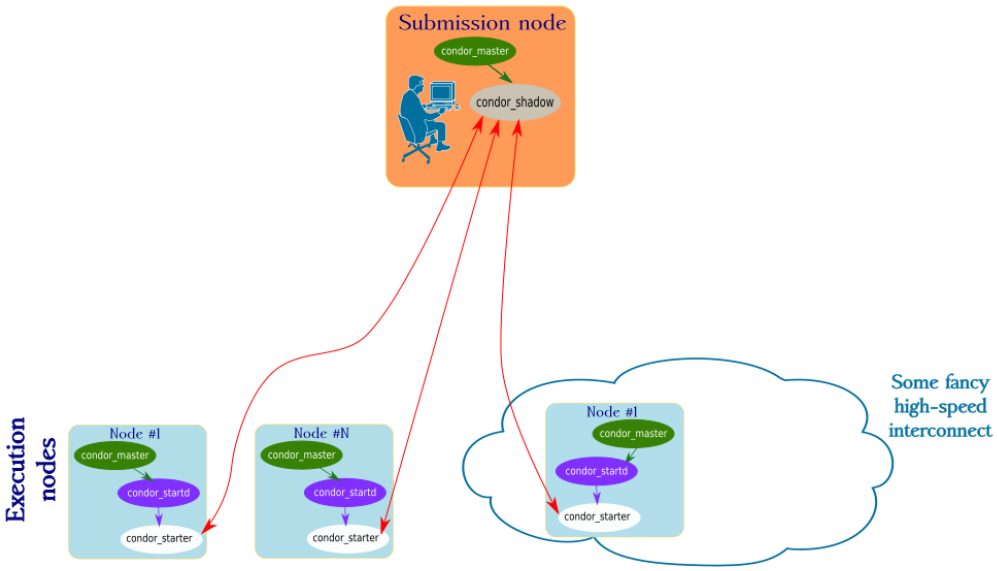
Startd flocking (1)



Startd flocking (2)



Startd flocking (3)



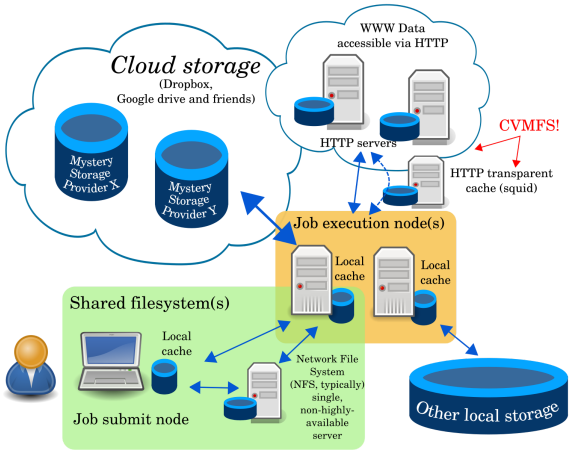
Count (or compute) on 'AMICO'

Original picture bitmap source:www.publicdomainpictures.net.

License: CC0 Public domain

- So, we eventually got to federate together all the different group-owned and operated clusters, with management backing...
- ... but no funding other than a few fractions of people willing to take part in the enterprise.
  "Nobody can beat us on the price".
- In order to look more real, we put together a name for the project and a nifty logo!
- **A**pparato **MI**lanese per il **C**alcolo **O**pportunistico - 'Milanese' Apparatus for Opportunistic Computing
- Could as well be: **A MI**lan **CO**ndor ...? Thingy ?

The AMICO Storage Model



- While Condor takes care of the computing needs of the 'AMICO' cluster, a CEPH cluster is used to provide readable/writable storage.
- Normal access is via S3 and the RADOS gateway (via S3FS where FUSE-mounting is allowed).
- CVMFS (read-only) access is also provided across all clusters for the benefit of WLCG users (this required some backporting effort). CVMFS is also mounted by default inside Docker containers.

AMICO pool config (1)

Here is a reference configuration: using this with partitionable slots required a few patches related to the 'dollar-dollar' expansion. These went into Condor v8.7.5 and later.

```
NegotiatorName = "whatever-pool"
NEGOTIATOR_MATCH_EXPRS = NegotiatorName
SUPER_COLLECTOR = superpool-cm.fisica.unimi.it
LOCAL_COLLECTOR = $(CONDOR_HOST)

# the local negotiator should only ever report to the local collector
NEGOTIATOR.COLLECTOR_HOST = $(LOCAL_COLLECTOR)
# startds should report to both collectors
STARTD.COLLECTOR_HOST = $(LOCAL_COLLECTOR),$(SUPER_COLLECTOR)

# trust both negotiators
#ALLOW_NEGOTIATOR=$(COLLECTOR_HOST)
ALLOW_NEGOTIATOR = $(LOCAL_COLLECTOR),$(SUPER_COLLECTOR)

# Flocking to super-pool
FLOCK_TO = $(SUPER_COLLECTOR)
```

AMICO pool config (2)

```
# Advertise in the machine ad the name of the pool
ClusterName = $(NegotiatorName)
STARTD_ATTRS = $(STARTD_ATTRS) ClusterName

# Advertise in the machine ad the name of the negotiator that made the match
# for the job that is currently running.  We need this in SUPER_START.
CurJobPool = "$$(NegotiatorMatchExprNegotiatorName)"
SUBMIT_EXPRS = $(SUBMIT_EXPRS) CurJobPool
STARTD_JOB_EXPRS = $(STARTD_JOB_EXPRS) CurJobPool

# Turn PREEMPT on only for jobs coming from an external pool
PREEMPT = ($(PREEMPT)) && (MY.CurJobPool =!= $(NegotiatorName))

# We do not want the super-negotiator to preempt local-negotiator matches.
# Therefore, only match jobs if:
#      1. the new match is from the local pool
#   OR 2. the existing match is not from the local pool
SUPER_START = NegotiatorMatchExprNegotiatorName =?= $(NegotiatorName) || \
              MY.CurJobPool =!= $(NegotiatorName)

START = ($(START)) && ($(SUPER_START))
```

AMICO pool config (3) - start policy

- The main concern of all pool owners is that workload that is submitted locally (interactive, MPI, Condor, slurm, whatever) has priority over remote pool jobs, i.e. prevents remote jobs from matching, and suspends/evicts remote pool jobs.

- After a few rounds of trial&error, we settled on suspending jobs after 2 minutes of non-local-Condor load, and evicting them after 10 minutes (config in the context of `condor_examples/condor_config.generic`):

```
AcceptableJob = ((TotalCPUs - TotalLoadAvg) >= TARGET.RequestCpus)
RemoteJob = (MY.CurJobPool =!= $(NegotiatorName))
WANT_SUSPEND = ( ( $(SUSPEND) ) && (TARGET.AvoidSuspend =!= True) )
WANT_VACATE  = ( $(ActivationTimer) > 10 * $(MINUTE) || $(IsVanilla) )
START   =  (DynamicSlot == True) || ( $(AcceptableJob) || \
           (State != "Unclaimed" && State != "Owner") )
SUSPEND = ( (CpuBusyTime > 2 * $(MINUTE)) && $(ActivationTimer) > 180 ) && \
           ($(RemoteJob) == True)
CONTINUE = $(CPUIdle) && ($(ActivityTimer) > 10)
```

Reality check on workloads

- While the **standard** universe still serves nicely a number of customers who just run a single executable (usually compiled from Fortran source...) and benefit from static linking, remote I/O and checkpointing, more complex workloads are served by:
  1. Running Docker containers (we install Docker or have Docker installed wherever possible on the various pools, and use `HasDocker`).
  2. Mounting CERN's CVMFS for read-only access to common software distributions, (we publish a `HasCVMFS` attribute).
  3. Providing users with CEPH-based object storage space, and encouraging them to redirect all job I/O there.
  4. Waiting for CRIU to provide workable checkpoint&restore capabilities for Docker containers.

- As mentioned, many users run various flavours of MPI jobs. Local pools are configured with one Dedicated Scheduler each, and can in principle be shared by multiple groups, as long as the proper MPI environment can be set up.

Reality check on MPI workloads

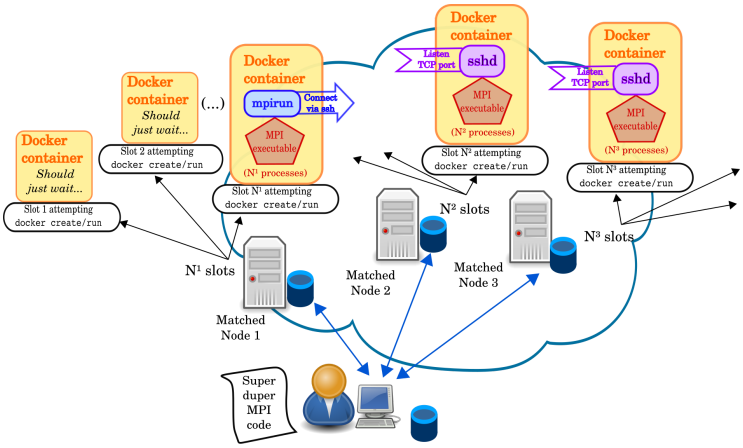
Roughly half of our reference users do use MPI, with little *a priori* or *a posteriori* awareness of the actual degree of paralellism in the code. To gain a better ability to serve their needs we selected three cases:

1. CRYSTAL, from the Turin Polytechnic, used to compute the ground-state wave function of periodic systems.
   - Based on OpenMPI
2. Locally-developed Shadow Path Integral Ground State (SPIGS) Monte Carlo code (used to compute the ground-state wave function of a quantum multi-body system) and Genetic algorithm via Falsification of Theories (GIFT) to compute e.g. Fourier inverse transforms.
   - Based on certain versions of MPICH, requiring Intel compilers
3. ADDA, to simulate light diffusion/scattering experiments of 3-d samples.
   - Based on OpenMPI.

Parallel Docker - 'paddock' (1)

- Can we actually *do* something for these MPI workloads on the 'AMICO' infrastructure?
- Is there some *goodput* that can be obtained, or is the execution performance orders of magnitude worse?
- Different versions of MPI, compilers and execution environments call for containers: we were able to assemble Docker containers with all the required dependencies to `mpirun` our local MPI applications.
- Sometimes it's hard to find enough slots on one physical node, so we attempt the **pa**rallel scheduling of **dock**er containers, hence the code-name **pa-(d)-dock**.
- → Via the use-at-your-own-risk `WantParallelScheduling = true` knob....
- This genuine Rube Goldberg, or swiss pinball machine can crank far enough to produce some performance measurements, so let's enter into a few details.

Parallel Docker - 'paddock' (2)



Main issues:

1. The handling of Docker Universe jobs can only be tweaked on the execution nodes by changing what the `DOCKER` config knob points to.
2. When N slots are requested on a given node, N `docker create/run` commands are attempted. MPI, however, wants to run N processes from a single entry point.
3. `condor_chirp` is probably the most valuable tool here, but the amount of reverse-engineering required into MPI implementations should be limited. The old SSHD mesh approach was more general than the current `orted_launcher`.

Parallel Docker - 'paddock' (3)

- So, 'paddock' is currently in the form of a set of BASH scripts meant to replace the real `'docker'` command in the `DOCKER` config variable. These take care of:
  1. Finding via an election cycle a range of available network ports that process running inside the Docker container can bind to (MPI implementations currently assume they can use the same ports on all cluster members).
  2. Making sure that only one slot per physical node attempts to actually run the requested Docker container (MPI will launch N processes from there). As a successful `docker create/start` command has to return a valid container ID, waiting/sleeping containers are created.
  3. Creating/starting the container, adding the relevant configuration parameters. Inside all containers but the master (process '0') a `sshd` will be started, and its access data reported via `condor_chirp`, like in the old version of the MPI support scripts.
  4. Handling signals and cleaning up as appropriate.
- Note: We needed to apply a few patches to Condor - all related to the use of `WantParallelScheduling = true`, which here and there doesn't get the same treatment of historic 'parallel' universe jobs.

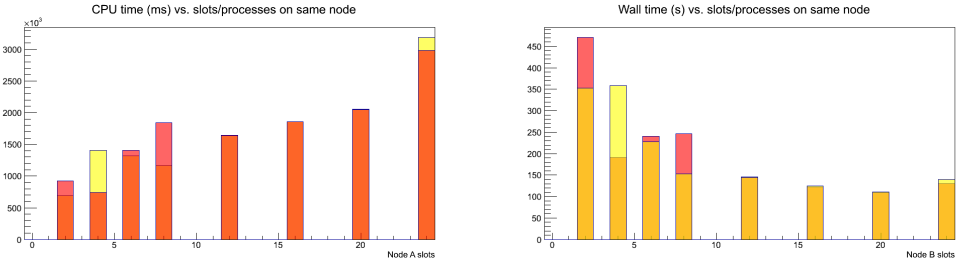
Sample submit file to run a 'paddock' job

Quiz: what of the following is *not* found in the Condor manual?

```
ıtParallelScheduling = true
ıverse = docker

ıuestmemory = 1024M

:ker_image = dr.mi.infn.it/run-crystal17-openmpi
ecutable = mpirun-exec-stats
ıuments = /path/to/MPPcrystal

ould_Transfer_Files   = YES
en_To_Transfer_Output = ON_EXIT_OR_EVICT
ansfer_Input_Files = INPUT,condor_chirp.x,\
                     wait-for-head-node
:put = run_crystal_stdout.$(Process).#pArAlLeLnOdE#
or = run_crystal_stderr.$(Process).#pArAlLeLnOdE#

ı = run_docker_log
```

```
#Enable chirp!
+WantIOProxy = true
+JobRequiresSandbox = true

+ParallelShutdownPolicy = \
          "WAIT_FOR_ALL"

machine_count = XXX
requirements = RXXX
queue
machine_count = YYY
requirements = RYYY
queue
```

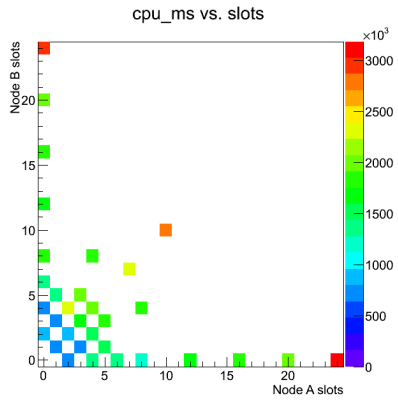
CRYSTAL runs on 'paddock' (1)

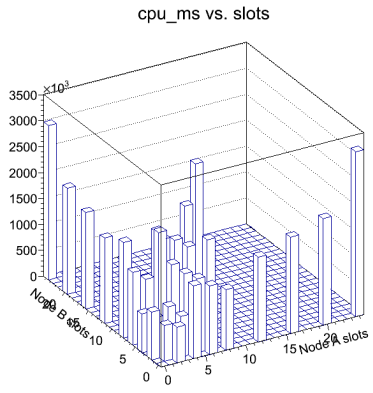


Black-box characterization of the CRYSTAL code with end-user-provided input and run conditions.
Tests based on two identical 64-slot (with hyperthreading) Intel® Xeon® E5-4610 v2@2.30GHz PCs.
Single-node parallel execution.

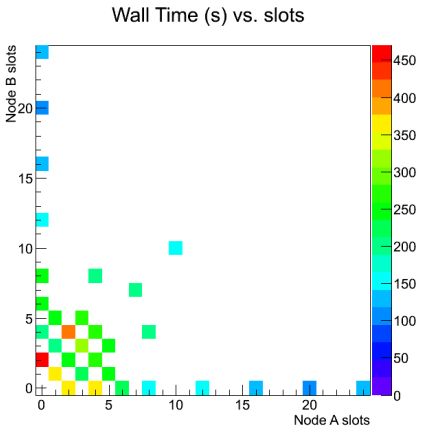
CRYSTAL runs on 'paddock' (2)

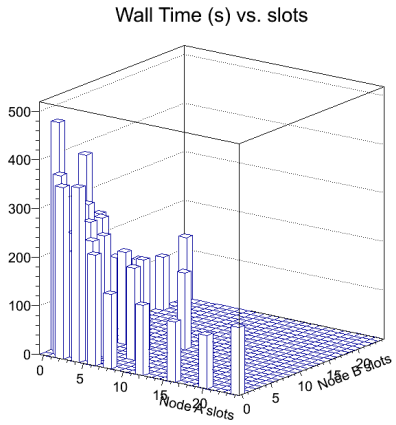
We then studied the execution of the CRYSTAL code by submitting 'paddock' execution on X slots on one physical node 'A' and Y slots on another physical node 'B' (with the MPI master is always on node 'B') - these are *all* preliminary results:



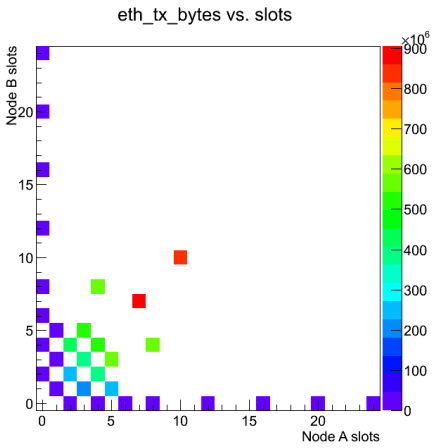


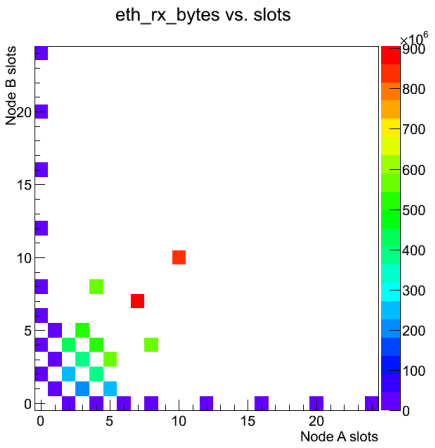
CRYSTAL runs on 'paddock' (3)



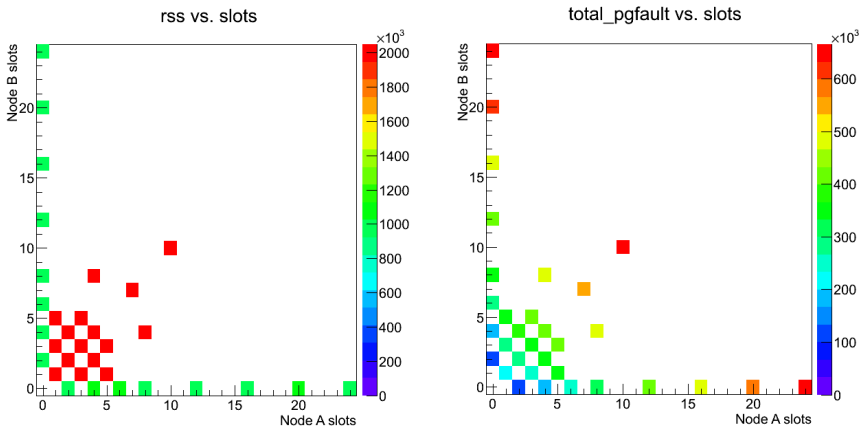


CRYSTAL runs on 'paddock' (4)



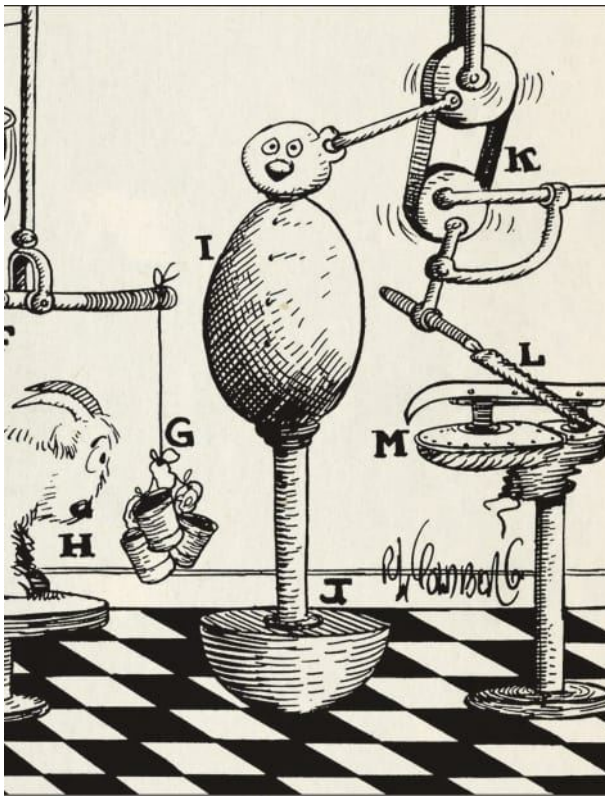


CRYSTAL runs on 'paddock' (5)



Conclusions

- The 'AMICO' set-up seems to be working, and to be friendly enough, as promised.
  - Sometimes it takes some persuasion to un-cling people from *their* own resources.
  - Most of the times user education/'facilitation' is needed.
- With enough persistence in interacting with the Condor devel team, order can be brought to cases where the configuration semantics doesn't produce the expected effects...
- We managed to coax a few reference MPI applications to run on the 'AMICO' Condor pools, via a Rube Goldberg assembly of Docker containers and parallel scheduling.
  - This required a few patches into Condor, that will eventually make it into the mainstream distribution.
- Systematic characterization of the MPI applications performance is on-going: the magnitude of the wall-time is comparable to single-node execution.



Questions

- Thank you for your time.

Questions

- Thank you for your time.