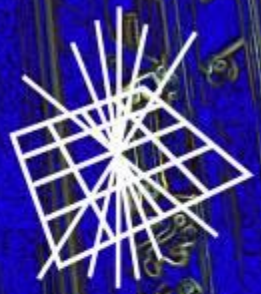


GridPP

UK Computing for Particle Physics

DrainBoss
A Drain Rate Controller for ARC/HTCondor

Introduction

This talk describes DrainBoss, which is a proportional integral (PI) controller with conditional logic that strives to maintain the correct ratio between single-core and multi-core jobs in an ARC/HTCondor cluster.

Steve Jones
University of Liverpool

Problem

- Consider a node with eight cores, running eight single core jobs. One is the first to end; a slot becomes free.
- But say the highest priority queued job needs eight cores.
- The newly freed slot is not wide enough to take it, so it has to wait.
- Should the scheduler use the slot for a waiting single core job, or hold it back for the other seven jobs to end?
- If it holds jobs back, then resources are wasted.
- If it runs another single core job, then the multicore job has no prospect of ever running.

Multicore jobs need all lanes clear at the right time

Condor's Solution

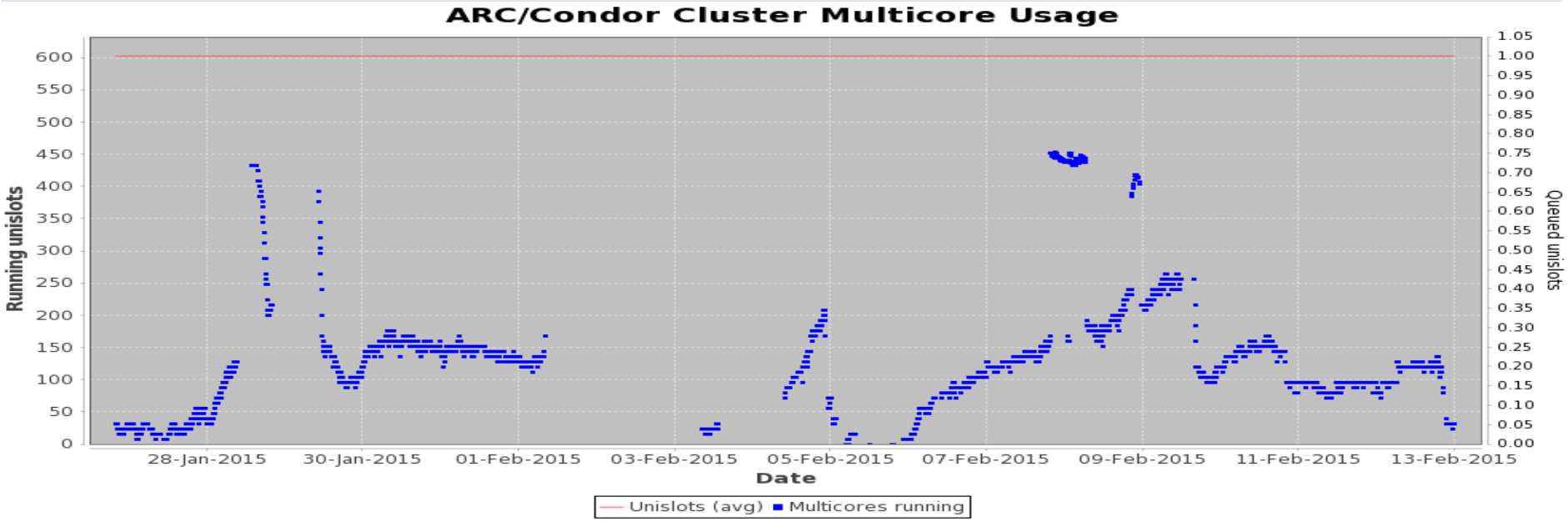
- The solution that Condor provides has two rules: periodically drain down nodes so that a multicore job can fit on them, and start multicore jobs in preference to single core jobs so they get on the newly drained nodes.
- This is implemented using the Condor DEFRAG daemon, and various job priority parameters. The daemon has parameters which control the way nodes are selected and drained for multicore jobs.

DEFRAG Daemon

- The version we use, 8.2.2, is good (less buggy)
- Main parameters:
  - MAX_CONCURRENT_DRAINING - Don't let more than this drain at once
  - DRAINING_MACHINES_PER_HOUR - Never start more than this many draining per hour
  - MAX_WHOLE_MACHINES - Don't bother draining if this many machines already have wide slot
- State constituting a WHOLE_MACHINE defined in an expression (classad)
- Tailor those constraints to get the drain rate you “want”; can be automated in (e.g.) cron.
- ClassAds very flexible for tailoring functionality, but they are not a “programing language”.

Daemon Performance

- Modifying the daemon parameters over a period of 2.5 weeks while collecting data showed:
- avg=121.82
- st. dev=63.07
- wastage: 5.21



Daemon Performance

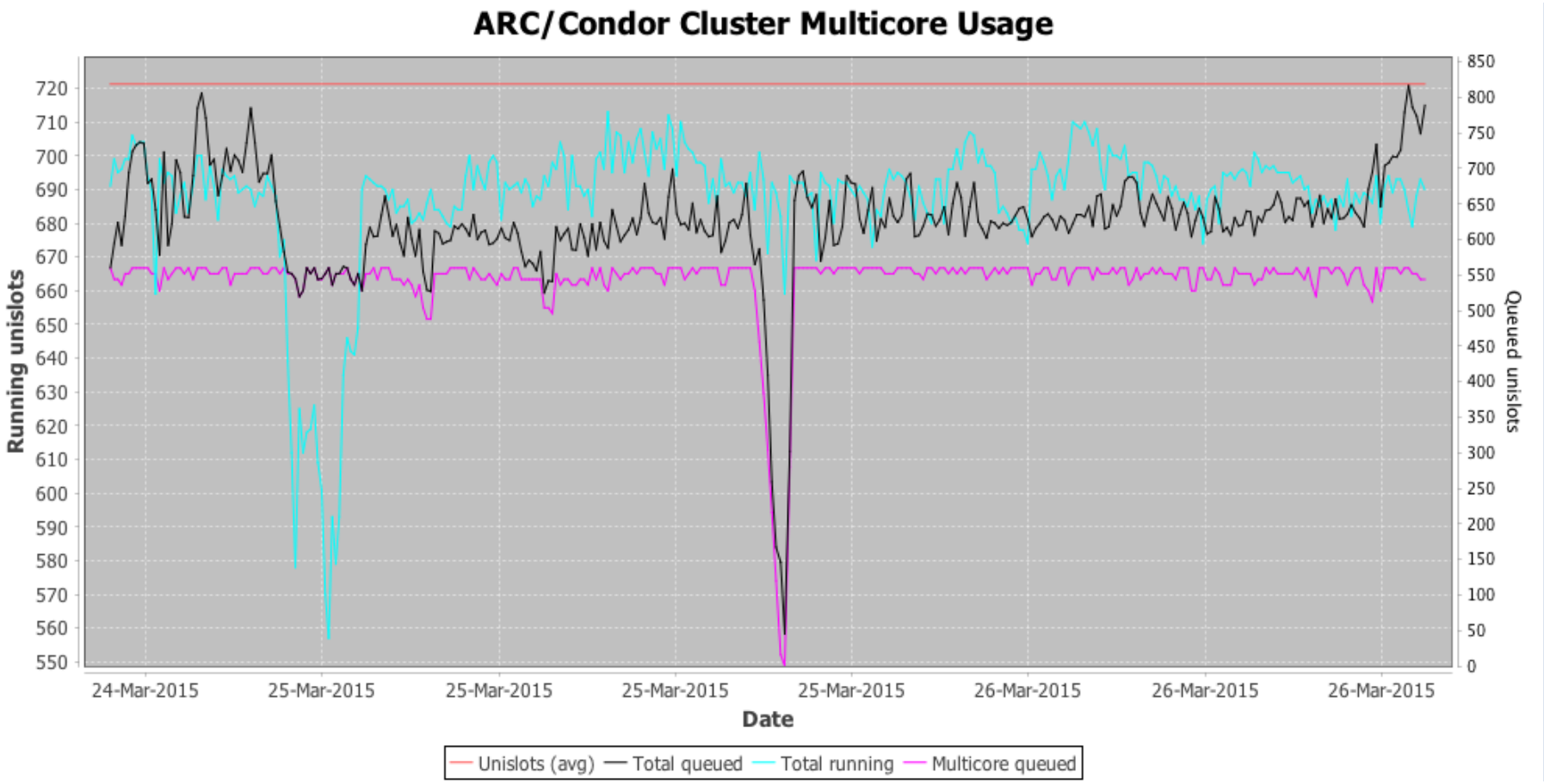
- It seems a bit scrappy...but…
- I didn’t do much systematic testing to establish a baseline.
- And I can't blame the daemon anyway – I was:
  - modifying it,
  - trying different rates,
  - different limits,
  - automatic adjustments and
  - the job traffic was sporadic.
- But I wondered what else is available?

Commercial "Solutions"

Feedback Ideas

- There is no off-the-shelf solution, but what if if we turn the problem around and found a way to tell the cluster “We want multicore jobs to use (say) 250 slots”? How could that be implemented.
- The choices appear to be either feedback or feedforward. A feedforward scheme would examine the traffic coming from upstream and try to make adjustments to account for it. A feedback loop looks at the state now and in the past and tries to guess what might happen in future based on that.
- Feedforward looked hard, while feedback seemed easy. But it doesn't work on random inputs. So are the inputs random?
- No. The next plot shows typical multicore and singlecore waiting jobs at our site. It’s being level controlled by something upstream, at ~ 600 jobs.

Waiting jobs



DrainBoss Principles

- It's not random, so a home-made feedback controller might work.
- The objectives are to maximise the usage of the cluster and get good mix of both single-core and multicore jobs by striving to obtain good control when submission is ideal, but not cause harmful effects when submission deteriorates.
- It has a process controller which senses condition of cluster and adjusts how nodes are drained and put back to obtain a certain amount of predictability.
- It has simple state logic to try to minimise negative corrections and deal with irregular delivery of multicore and single core jobs.
- It also needs a mechanism to start multicore jobs in preference to single core jobs.
- The prototype is implemented as a script (drainBoss.py) not a daemon.

Controller Principles

- The process controller provides the feedback control system.
- It measures some variable, and finds the error compared to some setpoint.
- Then it corrects the process to eliminate the error.
- DrainBoss uses Proportional and Integral terms.
- Proportional term (gain) acts proportionally to the error.
- Pure proportional control is sensitive to long time lags.
- Integral action sums the error over time; output grows to offset error.
- Proportional part + integral part eventually overcomes the error, I hope.

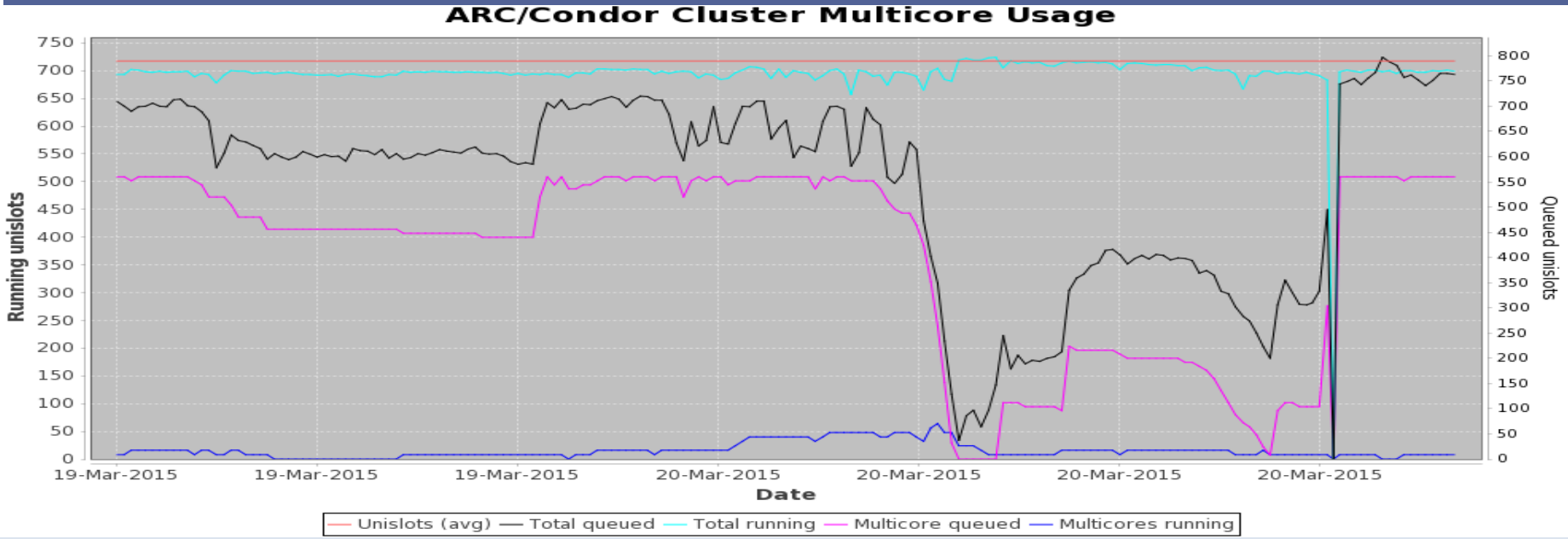
State Logic Principles

Queue state				
Mc jobs queues	No	Yes	No	Yes
Sc jobs queued	No	No	Yes	Yes
Action:				
Start drain if nec.	No	Yes	No	Yes
Cancel current drains	No	No	Maybe	No

State Logic Justification

- No constant stream of mc and sc jobs jobs; if no multicores queued, then don't start any draining - no jobs to fill the slots.
- Don't stop drains early (1 exception). Drains are a cost, and cancelling throws away “achievement”. Drains are left to finish, in case multicore jobs come along soon.
- But: if there are no multicores but some singlecores queued, option to cancel on-going drains, otherwise singlecores would be held back for “no valid reason” violating the objective to maximise usage. Maybe a singlecore bird in the hand is worth two multicore birds in the bush?

The price of a bush bird



- Draining on 19th March to free mc slots after drought. Early on 20th, a short mc drought occurred, sc jobs still queued.
- So DrainBoss cancelled all draining, because a bird “in the hand...”. Hm... now we have to wait another long time.
- Result: option added called --keepgoing

Parameters

```
# ./drainBoss.py -h
This program controls the drain rate on a condor server
using a process controller.

The options to this program are:
 -s  --setpoint  250     the setpoint value
 -p  --propband  200     proportional band
 -r  --reset     10000   reset time
 -l  --lookback  86400   look back time
 -m  --maxtodrain 9      max that can drain at once
 -t  --test              test mode
 -k  --keepgoing         keep going, don't cancel draining
```

Parameters

-s 250	The setpoint, telling the controller to try to keep 250 multicore jobs running.
-p 750	The proportional band. This is a wide band, greatly limiting effect of proportional term.
-r 43600	The “integral time”, which controls the importance of the accumulated error in the final correction. Used in denominator, so bigger number makes accumulated error less important.
-- lookback 86400	How far back to look at accumulated error, to avoid windup.
--maxtodrain	Extent of controller output; maximum size of correction (minimum is zero).
--keepgoing	Do not cancel drains even when zero multicores while singlecores queued.

Running it

```
#!/bin/bash
while [ 1 ]; do
  date;
  ./drainBoss.py -s 250 -p 750 -r 43600 \
       --lookback 86400  --maxtodrain 7 --keepgoing
  sleep 300;
done >> drainBoss.log
```

Starting and stopping draining

- Each time drainBoss runs, it potentially starts and stops drains.
- Starting drains: n nodes are selected by randomising the list of nodes and selecting the first n from the list that:
- are not not draining and
- have no slot composed of 8 or more “unislots”.
- Stopping drains: Each draining node that has any slot (used or free) composed of 8 or more “unislots” is put back in use.
- Thus the cluster is (almost) limited to max of one multicore job per node.

Preferring multicore jobs

- No matter how much we drain, if the system prefers singlecore over multicore, the multicore will not get scheduled.
- Even if mc and sc are equal, risk that “achievement” after draining is thrown away if (say) one sc spoils the newly drained node.
- Need to systematically prefer multicore to achieve objective to maximise the usage of the cluster.
- Tried several ways, inc.
- Raise the user priority of multicore jobs.
- Setting the GROUP_SORT_EXPR.

Preferring multicore jobs

- Raise the user priority of multicore jobs; brutally effective, using a cron job that finds mc jobs and runs "condor_userprio jobno -setfactor 250"
- GROUP_SORT_EXPR; Needs accounting groups. This setting seemed to work OK for a while by preferring High Priority and test/ops jobs, then mc jobs, and sc jobs last:

```
GROUP_SORT_EXPR = ifThenElse(AccountingGroup=?="<none>",
3.4e+38, ifThenElse(AccountingGroup=?="group_HIGHPRIO", -23,
ifThenElse(AccountingGroup=?="group_DTEAM", -18,
ifThenElse(AccountingGroup=?="group_OPS", -17,
ifThenElse(regexp("mcore",AccountingGroup),
ifThenElse(GroupQuota > 0 && GroupResourcesInUse > 0, (-1 *
GroupQuota) / GroupResourcesInUse ,-1),  ifThenElse(GroupQuota
> 0, GroupResourcesInUse/GroupQuota, 3.2e+38))))))
```

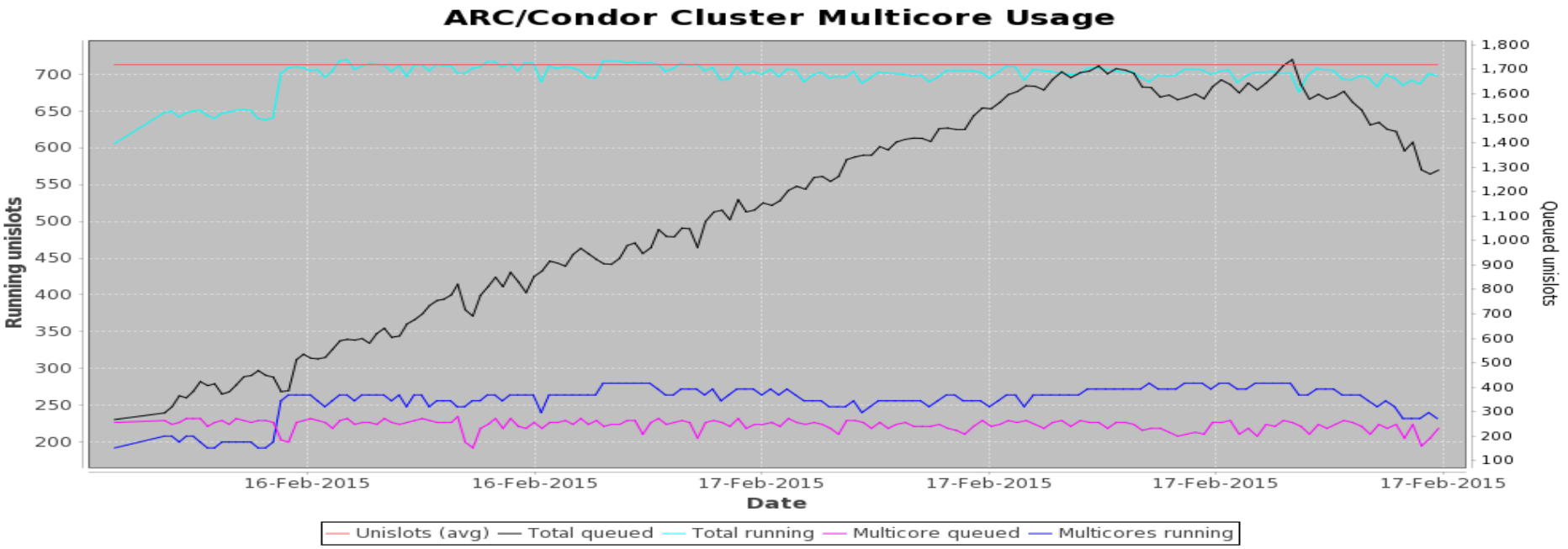
WARNINGS

- The GROUP_SORT_EXPR works in an opposite manner to how it is described in the manual for version 8.2.2. So smaller numbers = higher priority in the sort.

- Needs to be tuned; tuning was done by hand although there are supposedly technical ways to tune these PI systems more accurately that I hope to look at in future.

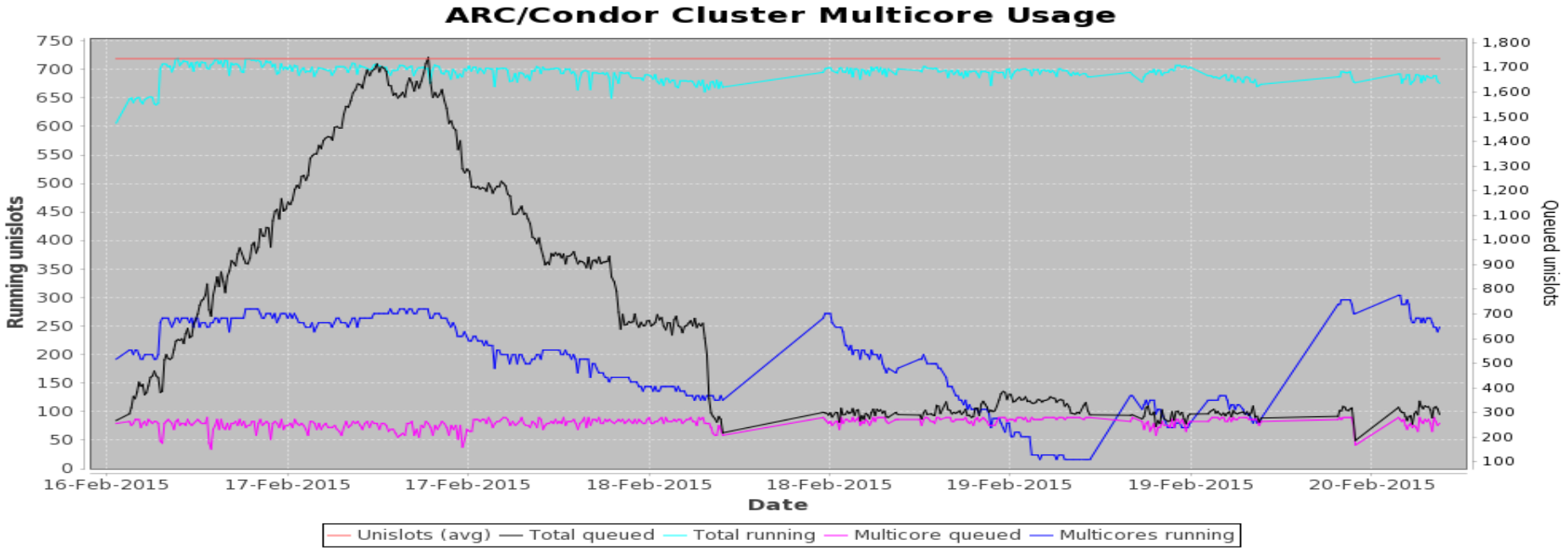
Performance

- I'll show some plots of the performance of the controller that cover interesting periods.
- I'll show it “warts and all”, but I'll compare the performance with a time-line of changes that partially explain some of the observations
- With such large variations, it's hard to be sure that it works, let alone whether it works better than an open loop approach.
- But time will tell.



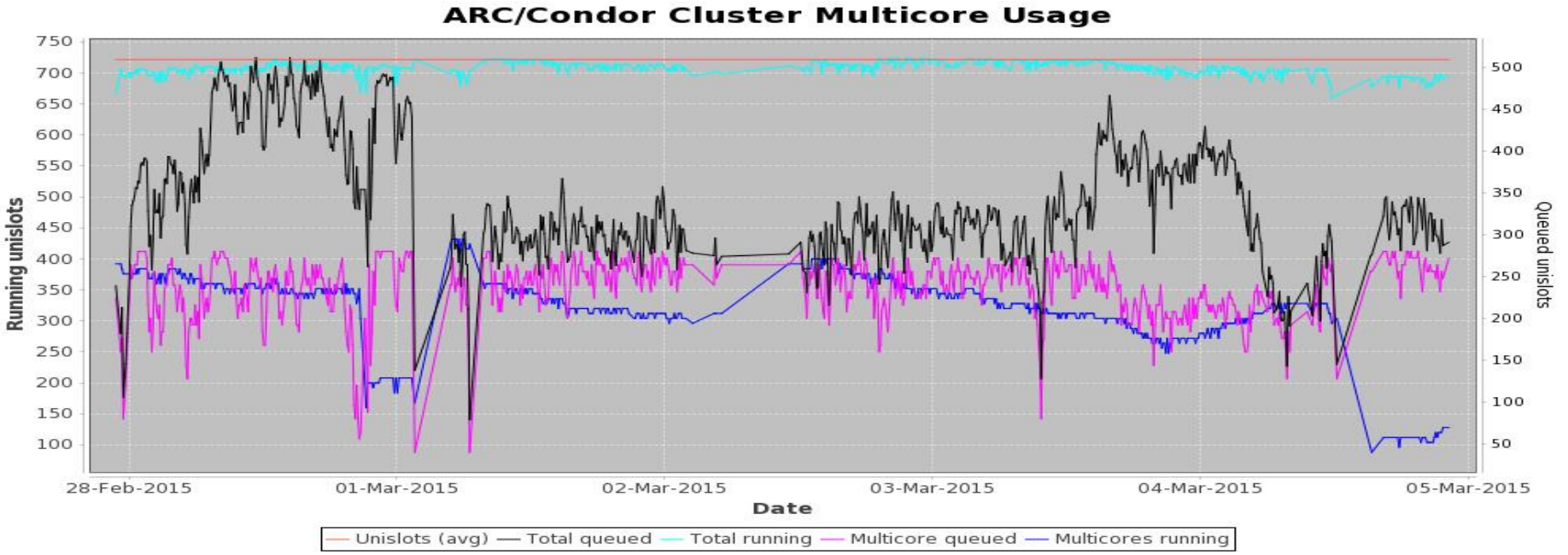
The proportional controller was started 16th Feb. The plot shows a stretch of apparently good control. But it doesn't last.

Second Glance



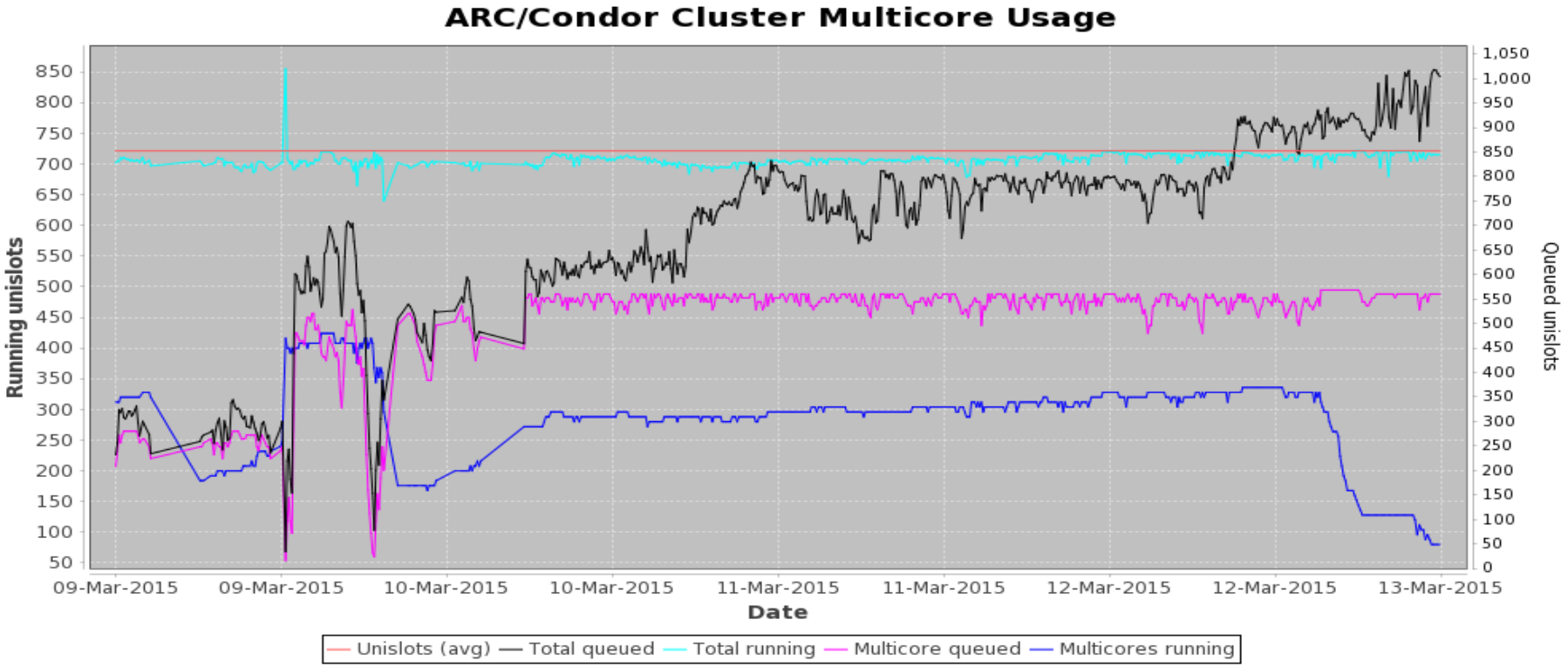
It was a mirage. In the bigger picture, the control deteriorates. It hunts around like this until 23rd, when I put in the integral term, which I tune for a few days.

Integral Action



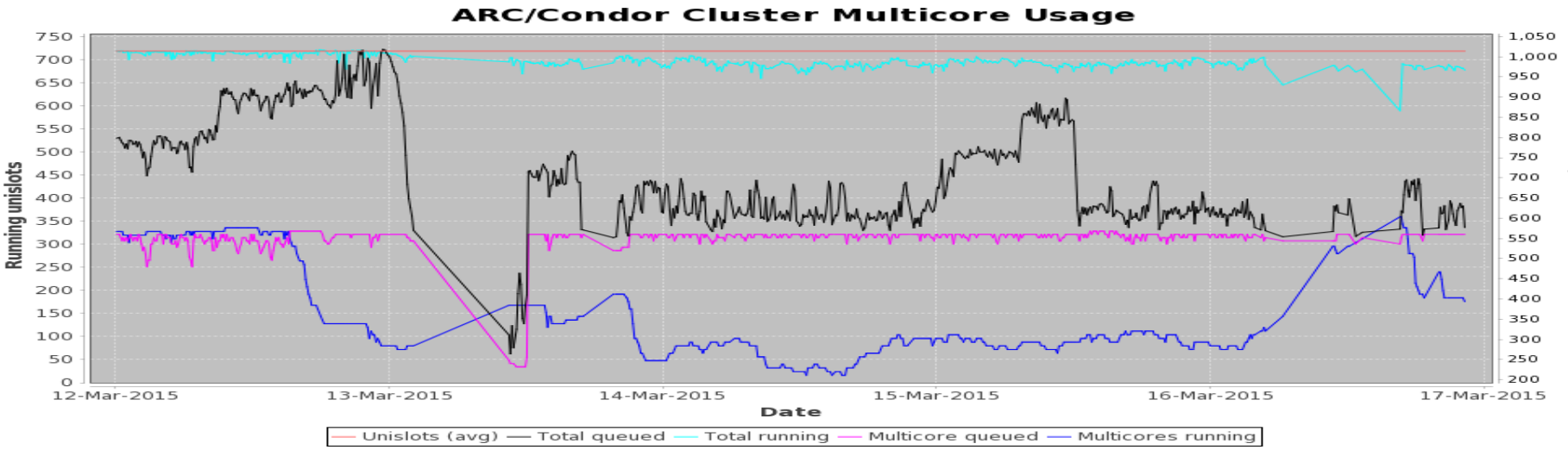
Once tuned, it seemed to control (with an offset) up until the 5th, when the submission system became too irregular.

Integral Action



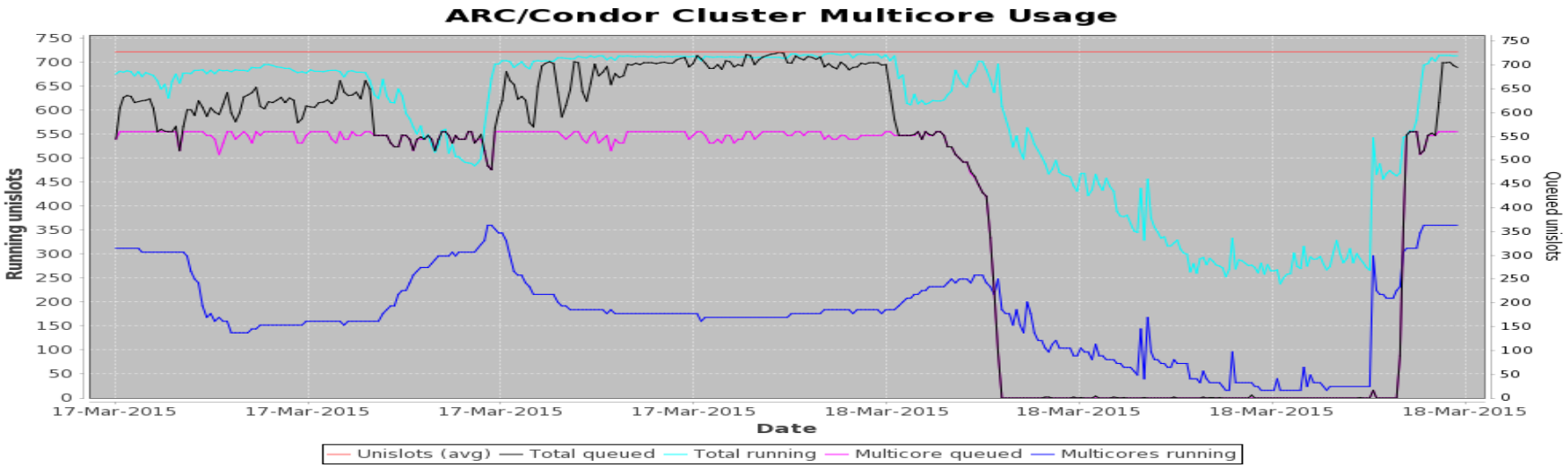
The submissions improved around the 9th. I intervened on the 12th to try to reduce the control offset.

But if picked a poor setting



- But I chose poor settings. This may be down to a misunderstanding about GROUP_SORT_EXPR which I corrected on the 16th.

After the fix



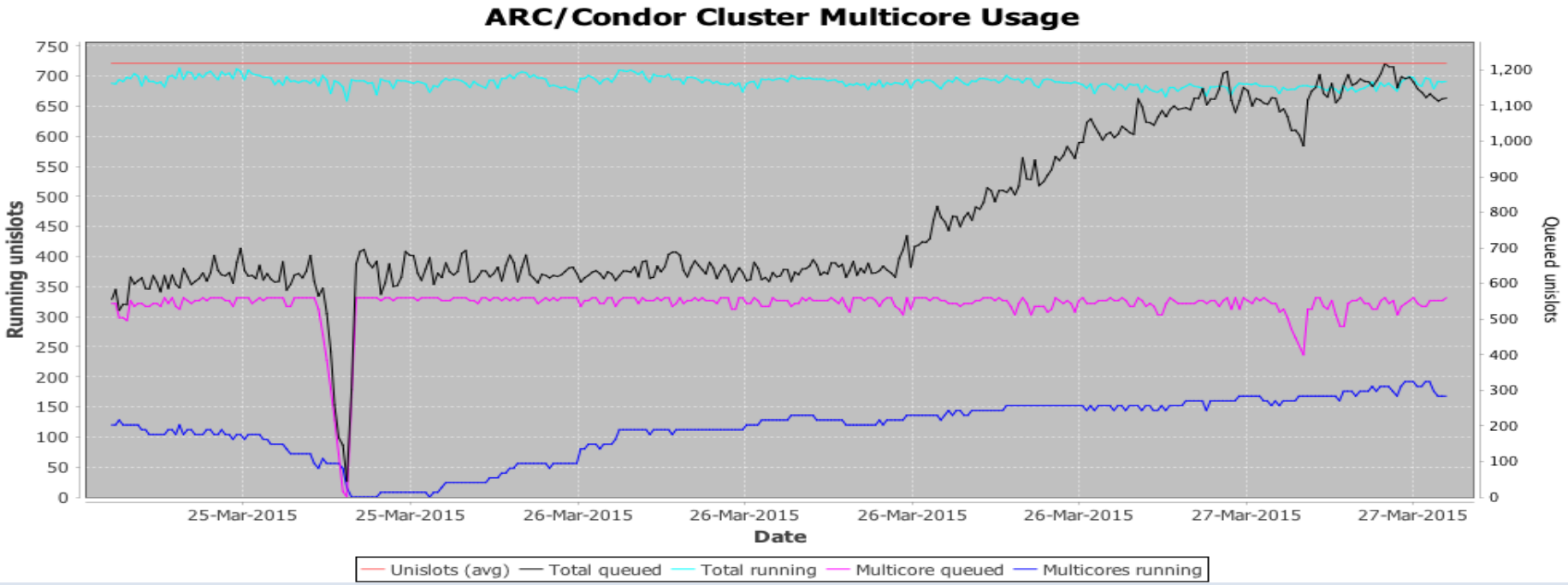
- Newer data shows the controller slowly recovering. The submissions deteriorate on the 18th.
- Note: this plot shows data during a poor submission period; it was omitted for clarity in the earlier plots.

Wastage

Qualification: I omitted data during periods where the submission system delivered no multicore jobs – you can't blame the controller for a job drought. And I have omitted data between the 12th and 16th of Feb, when a poor GROUP_SORT_EXPR setting was used.

- avg=298.61 (versus 121.82)
- st. dev=71.44 (versus 63.07)
- wastage - 2.43 (versus 5.31)

How's it doing this morning



- A bit low but moving in the right direction after a job drought 2 days ago.
- Emphasises need for a ramp up function when process is restarting.

Further work needed

- Port to other batch systems; e.g. torque.
- Error handling (it ignores them now)
- Ramp up function (PID controllers usually have them)
- Better selection of node to drain (largest is best)
- Integrate into CONDOR system, e.g. internal data structures
- Make into daemon, with clock to set run period.
- Much more systematic testing and tuning.
- Tuning guidelines.
- Release visualisation tools.

Conclusions

- Promising results:
  - Inputs not random.
  - Control can be achieved with good job delivery and payload pickup.
- Problems:
  - Erratic jobdelivery or poor payload pickup spoil things.

- I haven't seen anything to show the controller is worse than the DEFRAG daemon.
- That's faint praise, I know, but it's just a prototype.
- The wastage while it operates is low.
- It still has an offset that I haven't tried to explain yet.
- Overall I expect we'll keep using it unless something drastic happens.

Finally

- The program is here:

http://hep.ph.liv.ac.uk/~sjones/drainBoss.py

- The manual is/will be here:

https://www.gridpp.ac.uk/wiki/Example_Build_of_an_ARC/Condor_Cluster

- My email is sjones@hep.ph.liv.ac.uk

- Thanks are due to A. Lahiff (RAL) for several ideas and suggestions.
- Have a safe journey home.