ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Bologna

INFN/TC-03/18
11 Dicembre 2003

CONDOR AND THE BOLOGNA BATCH SYSTEM

Peter Couvares[1], Paolo Mazzanti[2], Daniela Bortolotti[2], Franco Semeria[2]

[1]*Department of Computer Science,University of Wisconsin-Madison, USA*
[2)]*INFN-Sezione di Bologna and Department of Physics, University of Bologna, Italy*

Abstract

A very simple batch system for UNIX Linux machines, based on Condor software is presented. This system may be very useful for managing farms and is very customizable and flexible. It has been extensively tested on a dedicated farm in INFN Bologna.

Published by ***SIS–Pubblicazioni***
Laboratori Nazionali di Frascati

1 INTRODUCTION

The Bologna Batch System is a custom batch scheduling system implemented on a small subset of machines within the larger nationwide INFN Condor pool. We used Condor's flexible policy mechanisms to specify the wishes of specific resource and job owners in Bologna in order to allow the batch dedicated machines to remain in the larger INFN pool, while prioritizing and managing jobs differently than elsewhere in the pool.

The Bologna Batch System policy seeks to balance the needs of special short- and long-running vanilla batch jobs, while allowing other Condor jobs to run only when none of these special batch jobs are waiting. Moreover, it seeks to prioritize explicitly-marked short-running batch jobs (one hour of wall-clock time or less) so that they are never kept waiting by long-running batch jobs, and also to enforce execution time limits so they cannot exceed their time. Batch servers should always run local batch jobs, regardless of other system load or user activity, but should run other Condor jobs only when the system is otherwise idle and there are no batch jobs waiting. Once started, neither long or short batch job may be preempted for other jobs.
This paper will explain the policy in detail, and how it was implemented with the Condor system.

2 POLICY AND REQUIREMENTS

The full policy, broken down into discrete requirements, is as follows:

1. Bologna Batch Jobs are specially-submitted jobs which may only run on specially-designated Bologna Batch Servers.
2. Only users in Bologna may submit Bologna Batch Jobs.
3. Bologna Batch Jobs must be vanilla-universe jobs (and therefore are not capable of checkpointing and resuming), and once they start they must not be preempted for other jobs.
4. Bologna Batch Servers prefer Bologna Batch Jobs over other Condor jobs, and will start Bologna Batch Jobs regardless of system load or console activity.
5. There are two types of Bologna Batch Jobs, short-running and long-running. Jobs are assumed to be short-running unless they are explicitly labeled as long-running when they are submitted.
6. A short-running Bologna Batch Job must not be forced to wait for the completion of a long-running Bologna Batch Job before starting.
7. When short and long-running Bologna Batch Jobs are running simultaneously on the same physical machine, the short-running job processes should run at a lower (better) OS priority than the long-running jobs.
8. A short-running Bologna Batch Job may only run for 1 hour, after which point it should be killed and removed from the queue.
9. Bologna Batch Jobs have priority over other jobs. This means two things: other jobs must never preempt Bologna Batch Jobs, and Bologna Batch Jobs must always immediately preempt other jobs.
10. Jobs should be spread among machines, in an attempt to achieve a reasonable load balancing.

3 POLICY IMPLEMENTATION

In order to implement this policy, we used the standard Condor policy expressions which dictate the wishes and requirements of resource and job owners in the Condor system.

First, we implemented policy requirement #1, *"Bologna Batch Jobs are specially-submitted jobs which may only run on specially-designated Bologna Batch Servers."*:

- We advertised the Bologna Batch Servers by adding a new BolognaBatchServer attribute to their startd classad ! Here we should add a footnote on ClassAdd or a pointer! This was accomplished by inserting the following two lines into each server's local Condor configuration file:

```
BolognaBatchServer  = True
STARTD_EXPRS = $(STARTD_EXPRS) BolognaBatchServer
```

- Similarly, users advertise Bologna Batch Jobs as such by placing a BolognaBatchJob attribute in their job classad. This is accomplished by inserting the following line into their job submit description files:

```
+BolognaBatchJob = True
```

- Once this has been done, and Bologna Batch Jobs and Servers can identify one another, users ensure that Bologna Batch Jobs run only on Bologna Batch Servers by specifying a job requirement. This is accomplished by inserting the following line into their job submit description files:

```
Requirements = (BolognaBatchServer == True)
```

To implement requirement #2, *"Only users in Bologna may submit Bologna Batch Jobs."*, the following mechanism was used:

- Each Bologna Batch Server double-checks the origin of a job claiming to be a Bologna Batch Job before believing it to be one. This is facilitated by defining the following shortcut macro in each server's local Condor configuration file:

```
IsBBJob  =  ( TARGET.BolognaBatchJob =?= True  && \ TARGET.SUBMIT_SITE_DOMAIN
== "bo.infn.it" )
```

(SUBMIT_SITE_DOMAIN is an attribute that INFN previously configured the Condor schedd to automatically add to each job's classad[1]. Individual Condor users are not able to override it.)

To implement policy #3, *"Only users in Bologna may submit Bologna Batch Jobs.*

---

[1] This is done by adding the following two lines to each server's local Condor config file:

```
SUBMIT_SITE_DOMAIN = "$(UID_DOMAIN)"
SUBMIT_EXPRS = $(SUBMIT_EXPRS) SUBMIT_SITE_DOMAIN
```

Bologna Batch Jobs must be vanilla-universe jobs (and therefore are not capable of checkpointing and resuming), and once they start they must not be preempted for other jobs.", the following mechanism was used:

- First, since all Bologna Batch Jobs must be vanilla universe jobs, each Bologna Batch Server will also double-check the universe of a job claiming to be a Bologna Batch Jobs before believing it to be one. This is accomplished by defining a IsVanillaJob shortcut macro and adding it to the IsBBJob expression we just created:

```
IsVanillaJob = (TARGET.JobUniverse =?= 5)

IsBBJob = ( TARGET.BolognaBatchJob =?= True && \
TARGET.SUBMIT_SITE_DOMAIN =?= SUBMIT_SITE_DOMAIN  && $(IsVanillaJob)  )
```

- Next, we modified each Bologna Batch Server's WANT_SUSPEND_VANILLA and PREEMPT expressions, which Condor uses to decide when to suspend or preempt a vanilla job, so that INFN's default preemption policy would only affect non-Bologna Batch Jobs. (We also added a simple shortcut macro to make it easy to refer to non-Bologna Batch Jobs.)

```
IsNotBBJob = ( $(IsBBJob) =!= True )

WANT_SUSPEND_VANILLA = ( $(IsNotBBJob) && ($(WANT_SUSPEND_VANILLA)) )

PREEMPT = ( $(IsNotBBJob) && ($(PREEMPT)) )
```

To implement policy #4, "*Bologna Batch Servers prefer Bologna Batch Jobs over other Condor jobs, and will start Bologna Batch Jobs regardless of system load or console activity.*", the following mechanism was used:

- We modified each BB server's RANK and START expressions as follows:

```
RANK = $(IsBBJob)

INFN_START = ( (LoadAvg - CondorLoadAvg) < 0.3 && KeyboardIdle > (15 * 60)  && \
TotalCondorLoadAvg <= 1.0 )

START = ( $(IsBBJob) || ($(INFN_START)) )
```

(The INFN_START expression simply contains the default Condor START policy used for non-BB jobs).

To implement policy #5, *"There are two types of Bologna Batch Jobs, short-running and long-running. Jobs are assumed to be short-running unless they are explicitly labeled as long-running when they are submitted.*", we did the following:

- Users label long-running jobs by placing a special attribute in their job classad. This is accomplished by placing the following line in their job submit description files:

```
+LongRunningJob = True
```

- Bologna Batch Servers then identify long and short-running jobs by looking for the presence or absence of this attribute, along with the IsBBJob attribute defined earlier:

```
IsLongBBJob = ( $(IsBBJob) && TARGET.LongRunningJob =?= True )

IsShortBBJob = ( $(IsBBJob) && TARGET.LongRunningJob =!= True )
```

To implement policy #6, "*A short-running Bologna Batch Job must not be forced to wait for the completion of a long-running Bologna Batch Job before starting*.", we did the following:

- In order to guarantee this, we assigned each Condor Virtual Machine (VM) in the Bologna Batch System a predefined role of running either long-running jobs or short-running jobs – in a sense, creating a separate group of dedicated resources for each type of job. This way, long-running jobs need never be preempted to start short-running jobs, nor will they delay the start of short-running jobs, because the two types never compete for the same VM. The downside of this approach by itself, of course, is the risk of unnecessarily idle resources. If all the short-running VMs are busy while long-running VMs are idle, and short-running jobs are waiting to run, available resources are being wasted. To eliminate this, we configured *more total Condor VMs on Bologna Batch Servers than there are actual CPUs*. For example, on a dual-CPU server we configured six Condor VMs (two for short-running jobs, and four for long-running jobs), and on a single-CPU machine we configured two Condor VMs (both for short-running jobs). Although this tactic may lengthen the wall-clock time it takes for individual jobs to complete if all VMs are busy, it should not hurt overall *throughput* substantially[1]. To accomplish all of this, we first defined expressions to represent the two different types of VMs, IsShortRunningVM and IsLongRunningVM.
- On machines with only one VM, we simply added lines like the following to the server's local Condor config file:

```
IsShortRunningVM = True

IsLongRunningVM = False
```

- On multi-VM machines, the expressions are based on how many of each VM type we wanted, like so:

```
NUM_SHORT_RUNNING_VMS = 2

IsShortRunningVM = (VirtualMachineID <= $(NUM_SHORT_RUNNING_VMS))

IsLongRunningVM = (VirtualMachineID > $(NUM_SHORT_RUNNING_VMS))
```

[1] Obviously, the OS overhead of multitasking incurs some cost, but assuming there is adequate physical memory for all running jobs, it should be minimal. Multitasking may even *help* overall throughput by increasing CPU utilization if individual jobs do not each utilize 100% of the CPU.

- Next, we ensured that each VM would only run jobs of the corresponding category by defining different START expressions for each type:

```
SHORT_RUNNING_VM_START = ( $(IsShortBBJob) || \
( $(IsNotBBJob) && $(INFN_START) ) )

LONG_RUNNING_VM_START   = $(IsLongBBJob)
```

(We decided that only short-running VMs would start non-BB jobs, so that the number of such jobs would not be allowed to exceed the number of physical CPUs, unlike BB jobs.)

- Finally, we used the VM type attribute to tell the START expression to use the appropriate policy given the current VM:

```
START = ( ( $(IsShortRunningVM) && $(SHORT_RUNNING_VM_START) ) \
|| ( $(IsLongRunningVM) && $(LONG_RUNNING_VM_START)  ) )
```

Note that this is not the same START expression that we used in step four, but is an evolution of that START expression. This start expression has one complication. Condor will only consider starting jobs if the machine is not in the Owner state. Condor decides if a machine is in the Owner state by evaluating IsOwner. The default expressions for IsOwner is:

```
IsOwner = !START
```

Because our expression refers to jobs and not just the state of the machine, this is inappropriate. Because there is no job available when evaluating IsOwner, the IsOwner expression effectively becomes:

```
IsOwner = !($INFN_START)
```

This means that no jobs can start until the machine is idle, but we wish short BB jobs to start immediately, whether or not the machine is idle. A simple solution is to use:

```
IsOwner = False
```

In this case, Condor will never enter the Owner state, and will just use the START expression to decide when jobs can start.

To implement policy #7, *"When short and long-running Bologna Batch Jobs are running simultaneously on the same physical machine, the short-running job processes should run at a lower (better) OS priority than the long-running jobs.*", we did the following:

- Taking advantage of the dynamic policy of the Condor startd, we specified that short-running BB jobs should run with OS priority 5, and BB jobs which are marked as long-running, along with all non-BB jobs, should be run at OS priority 15. To accomplish this, we defined the JOB_RENICE_INCREMENT expression in terms of

the LongRunningJob and BolognaBatchJob job attributes as follows, in each server's local Condor config file:

```
JOB_RENICE_INCREMENT = (5 + ( 10 * ( LongRunningJob =?= True\
|| BolognaBatchJob =!= True ) ) )
```

If LongRunningJob is true in the job classad, the expression evaluates to (5 + (10 * 1)), or 15. If LongRunningJob is undefined or false in the job classad, but BolognaBatchJob is true, the expression evaluates to (5 + (10 * 0)), or 5. If neither is defined, the expression evaluates to (5 + (10 * 1)), or 15.

To implement policy #8, *"A short-running Bologna Batch Job may only run for 1 hour, after which point it should be killed and removed from the queue.*", we did the following:

- First, we modified each batch server's PREEMPT expression to preempt short-running batch jobs at the appropriate time:

```
PREEMPT = (  ( $(IsNotBBJob) && ( $(PREEMPT) )  ) \
|| (  $(IsShortBBJob) && ( $(ActivityTimer) > 60*60)  )  )
```

- We also modified the SHORT_RUNNING_VM_START expression on each batch server so that jobs preempted in this way will not restart:

```
SHORT_RUNNING_VM_START = ( ( $(IsShortBBJob) &&  \
( RemoteWallClockTime < 60*60 )  =!=  False ) || ( $(IsNotBBJob) && ( $(INFN_START) ) ) )
```

- However, there is no way for the remote startd to remove a job from the user's queue, so jobs preempted in this manner would remain in the queue, unable to match with any resources. To address this, we take advantage of the Periodic_Remove expression evaluated periodically by the Condor schedd. To do so, each user adds the following attribute to their submit file:

```
Periodic_Remove = ( LongRunningJob =!= True && ( RemoteWallClockTime < 60*60 )  )
```

This tells the condor_schedd to remove the job from the queue as soon as this expression becomes true.This expression is true when the job is not marked as long-running, and has run for more than one hour.

To implement policy #9, *"Bologna Batch Jobs have priority over other jobs. This means two things: other jobs must never preempt Bologna Batch Jobs, and Bologna Batch Jobs must always immediately preempt other jobs.*", we did the following:

- There are two reasons that one job may be preempted for another job in Condor: to honor the machine's job preferences (i.e., startd RANK), or for reasons of user priority. Our earlier changes to the startd's RANK expression addressed the first case, but in order to address the second we modified the Condor pool negotiator's configuration file as follows:

```
PREEMPTION_REQUIREMENTS =  ( ( BolognaBatchServer =!= True && \
( $(StateTimer) > (2 * (60 * 60)) && RemoteUserPrio > SubmittorPrio * 1.2 ) ) || \
(BolognaBatchServer =?= True && ( BolognaBatchJob =!= True && \
( TARGET.BolognaBatchJob =?= True || ($(StateTimer) > (2 * (60 * 60)) && \
RemoteUserPrio > SubmittorPrio * 1.2 ) ) ) ) )
```

In English, this says that on non-Bologna Batch Servers, the default INFN preemption requirements will apply, and any job that has run for more than two hours can be preempted by a sufficiently higher-priority job. On Bologna Batch servers, however, only a non-batch job can be preempted for that reason, or because the new candidate job is a batch job.

To implement policy #10, *"Jobs should be spread among machines, in an attempt to achieve a reasonable load balancing."*, we did the following:

- Condor does not do load balancing, largely because it has been unnecessary in the past. This is because Condor usually runs just one job per CPU. Because we are now running more than one job per CPU in the BBS, we want to attempt to spread the load between the machines. Condor cannot enforce load balancing, but we can encourage it by encouraging users to have their jobs rank machines based on load. For short BBS jobs, we want users to use a rank expression like this:

  ```
  Rank = ((TotalLoadAvg<= 2)* 2) + ((TotalLoadAvg<= 1) * 2) + (VirtualMachineId == 1)
  ```

  This will prefer machines with low load. For two machines with similar loads, this will prefer VM1, and this will spread jobs across machines. Think of a system where there are no jobs running, and the load is 0 on each machine. Condor will prefer VM1 over VM2, so the jobs will be evenly distributed across the machines. As more jobs are submitted, jobs will prefer machines with lower load.

- For long BBS jobs, we want users to use a rank expression like this:

  ```
  Rank = ((TotalLoadAvg <= 2) * 4)  + ((TotalLoadAvg <= 1) * 4) \
  + ((VirtualMachineId == 3)*3)  + ((VirtualMachineId == 4)*2) +  (VirtualMachineId == 5)
  ```

  This is analogous to the rank for short BBS jobs, except that it spreads the jobs across four virtual machines, not two. Fortunately, we do not need to merely request that users use these rank expressions. See the addendum below.

4 ADDENDUM

In order to facilitate users submitting Bologna Batch jobs, we created *bbs_submit_short* and *bbs_submit_long* scripts to automatically add all of the appropriate attributes to their job description before submission. The *bbs_submit* scripts call condor_submit with additional arguments adding each necessary Bologna Batch System attribute, and with a special configuration environment variable that modifies the users' requirements expression without overwriting it. The *bbs_submit_short* script looks as follows: (The rank is all one on line, even though it is wrapped in this version.)

```
CONDOR_APPEND_RANK_VANILLA  =  '( ( TotalLoadAvg <= 2 ) * 4 ) + \
( (TotalLoadAvg <= 1 ) *4 ) + ( ( VirtualMachineId == 3 ) *3 ) + \
( ( VirtualMachineId == 4 ) *2 ) + ( VirtualMachineId == 5 )'
_CONDOR_APPEND_REQ_VANILLA = '( BolognaBatchServer == True )'
export _CONDOR_APPEND_REQ_VANILLA
condor_submit -a '+BolognaBatchJob = True' \
-a 'should_transfer_files = IF_NEEDED'  \
-a 'when_to_transfer_output = ON_EXIT'  \
-a 'universe = vanilla'  \
-a 'periodic_remove = ( LongRunningJob =!= True && (RemoteWallClockTime > 60*60) ) '  $*
```

To submit a simple Bologna Batch job, the user simply need to run *bbs_submit_short* or *bbs_submit_long* from a Bologna Batch Server. Here is a trivial example job submit file "foo.sub":

```
universe = vanilla
executable = /bin/ls
output = ls.out
error = ls.err
log = ls.log
queue
```

To submit it to the Bologna Batch System, run:

```
bbs_submit_short  foo.sub
```

See also http://www.bo.infn.it/calcolo/condor/use_bbs.html for more examples.

5 REFERENCES

http://www.bo.infn.it/calcolo/condor/
http://www.cs.wisc.edu/condor/