

Condor and the Bologna Batch System Revision 4.0 (April 2004)

Peter Couvares <pcf@cs.wisc.edu>
Paolo Mazzanti <Paolo.Mazzanti@bo.infn.it>
Daniela Bortolotti <bortolotti@bo.infn.it>
Franco Semeria <Franco.Semeria@bo.infn.it>

Summary

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 their machines to remain in the larger INFN pool, while prioritizing and managing local jobs differently than elsewhere in the pool.

In addition to prioritizing local jobs, the primary goal of Bologna's policy is to allow short jobs to start quickly without waiting for longer batch jobs to complete, while simultaneously preventing long-running jobs from starving or CPUs from remaining idle. One way to address this goal is through the use of Condor's preemptive scheduling¹. However, due to software and social constraints, many of Bologna's local jobs are not able to checkpoint, and thus cannot be preempted without sacrificing throughput. Given this constraint, we implemented a system whereby resources could be temporarily overcommitted in order to provide quick response for short jobs, while allowing longer jobs to run to completion without preemption.

Whether or not these specific policies are applicable in your environment, we hope the discussion of their development and implementation below can help you realize the flexibility that Condor offers in terms of local control and policy configuration.

Details

The Bologna Batch System policy seeks to balance the needs of special short and long-running vanilla *Bologna Batch Jobs*, while allowing other Condor jobs to run only when none of these special Bologna Batch Jobs are waiting. Moreover, it seeks to prioritize explicitly-marked short-running Bologna Batch Jobs (requiring one hour of wall-clock time or less) so that they are never kept waiting by long-running Bologna Batch Jobs, and also to enforce execution time limits so they cannot exceed their time. Bologna resources should always run local Bologna Batch Jobs, regardless of other system load or user activity, but should only run other Condor jobs when the system is otherwise idle and there are no Bologna Batch Jobs waiting. Once started, neither long nor short-running Bologna Batch Jobs may be preempted for other Condor jobs.

This paper will explain the policy in detail, and how it was implemented with the Condor system.

¹ P. E. Krueger and Miron Livny, "A Comparison of Preemptive and Non-Preemptive Load Distributing", *Proc. of the 8th International Conference on Distributed Computing Systems*, pp. 123-130, June 1988.

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

1. Bologna Batch Jobs are specially-designated jobs which may run only 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 thus 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. Bologna Batch 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 one hour, after which point it should be killed and removed from the queue.
9. Bologna Batch Jobs have priority over other Condor jobs. This means two things: other jobs must never preempt Bologna Batch Jobs, and Bologna Batch Jobs must always immediately preempt other jobs.

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 #, *“Bologna Batch Jobs are specially-designated jobs which may run only on specially-designated Bologna Batch Servers.”*:

We advertised the Bologna Batch Servers by adding a new `BolognaBatchServer` attribute to their startd classad. 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 #, *“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 == $(SUBMIT_SITE_DOMAIN) )
```

(SUBMIT_SITE_DOMAIN is an attribute that INFN defines on all machines, and which they previously configured the Condor schedd to automatically add to each job's classad². Individual Condor users are not able to override it.)

To implement policy #, *"Bologna Batch Jobs must be vanilla-universe jobs (and therefore are not capable of checkpointing and resuming), and thus 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) )
```

(NOTE: In Condor's configuration macro language, the inclusion of `WANT_SUSPEND_VANILLA` in its own definition above is not recursive – rather it references the previous definition. In effect, we're simply adding to what was already defined, rather than redefining it entirely.)

To implement policy #, *"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 #, *"There are two types of Bologna Batch Jobs, short-running and long-running. Bologna Batch 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
```

² 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 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 #, “*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, all the long-running VMs are idle, and short-running jobs are waiting to run, then available resources are being wasted. To avoid 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 server 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* substantially³.

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))
```

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) ) )
```

³ 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 are not 100% CPU-bound.

To implement policy #, “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, while long-running BB jobs, 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 #, “*A short-running Bologna Batch Job may only run for one 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 short-running jobs preempted because they’ve run too long will not restart anywhere:

```
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 file⁴:

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

The `Periodic_Remove` expression tells the condor_schedd to remove the job from the queue if it ever becomes true. In this case, the expression becomes true when a BBS job is not marked as long-running, and has already run for more than one hour.

To implement policy #, “*Bologna Batch Jobs have priority over other Condor 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:

⁴ In fact, this attribute is automatically added on behalf of users; see Addendum #1 for details.

```

INFN_PREEMPTION_REQUIREMENTS =
  ( $(StateTimer) > (2 * (60 * 60)) \
    && RemoteUserPrio > SubmittorPrio * 1.2 )

PREEMPTION_REQUIREMENTS = \
  (( BolognaBatchServer!=True && $(INFN_PREEMPTION_REQUIREMENTS)) \
  || (BolognaBatchServer =?= True \
    && ( BolognaBatchJob != True \
      && ( TARGET.BolognaBatchJob =?= True \
        || $(INFN_PREEMPTION_REQUIREMENTS) ))) )

```

In English, this says that on non-Bologna Batch Servers, the default INFN preemption requirements will apply: 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-BB job can be preempted, and it can be preempted either for reasons of pool-wide user priority, or if the new candidate job is a BB job.

Addendum:

In order to make it easier for users to submit 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 for 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:

```
_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) ) ' \  
              $*
```

The *bbs_submit_long* script is identical, except for one added line, and one removed:

```
_CONDOR_APPEND_REQ_VANILLA='(BolognaBatchServer == True)'  
export _CONDOR_APPEND_REQ_VANILLA  
condor_submit -a '+BolognaBatchJob = True' \  
              -a '+LongRunningJob = 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) ) ' \  
              $*
```

Example:

To submit a simple Bologna Batch Job, a user simply needs to run *bbs_submit_short* or *bbs_submit_long* from a Bologna Batch Server. Here is a simple example job submit file:

```
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
```

Addendum #2:

After two years of experience with this system, we made additional enhancements to allow a specific workgroup within Bologna (the ALICE experiment) to add their resources to the Bologna Batch System (and in turn, to the INFN Condor pool) while guaranteeing that their jobs would never wait for other jobs (even other Bologna Batch Jobs) to complete before running on their own resources. In a sense, we replicated the local control that the Bologna Batch System exercises inside the wider INFN Condor pool *within* the Bologna Batch System itself.

We also took advantage of a new feature in Condor that allows one virtual machine to examine the state of all the other virtual machines running on the same server. This enabled us to modify the BBS policy to support the suspension of one type of BBS job while another BBS job runs, rather than simply setting their relative priorities in advance (as in the above document).

Specifically, we implemented the following policy for a subset of the Bologna Batch Servers, in addition to all the normal BBS policies implemented earlier:

10. There is a third type of Bologna Batch Job, an ALICE Job, which may run only on specially-designated ALICE BB Servers.
11. An ALICE Job must not be forced to wait for the completion of another Bologna Batch Job before starting on a ALICE BB Server.
12. ALICE BB Servers will not start long-running BB Jobs, but will start short-running BB Jobs when no ALICE jobs are running.
13. Short-running BB Jobs already running on an ALICE BB Server when an ALICE Job arrives will be allowed to run to completion, at a higher (worse) OS priority than the ALICE Job.

First, we implemented policy requirement #, “There is a third type of Bologna Batch Job, an ALICE Job, which may run only on specially-designated ALICE BB Servers.”

We advertised the ALICE BB Servers by adding a new `BolognaBatchGroup` attribute to their `startd` classads, in addition to the existing BB Server attributes. This was accomplished by inserting the following two lines into each server’s local Condor configuration file:

```
BolognaBatchGroup = "ALICE"  
STARTD_EXPRS = $(STARTD_EXPRS) BolognaBatchGroup
```

Similarly, users advertise ALICE Jobs as such by placing a `App` attribute in their job classad, in addition to the existing BB Job attributes. This is accomplished by inserting the following line into their job submit description files:

```
+App = "ALICE"
```

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

```
Requirements = (BolognaBatchGroup == "ALICE")
```

Next, we implemented policy requirement #, “An ALICE Job must not be forced to wait for the completion of another Bologna Batch Job before starting on a ALICE BB Server.”

In order to guarantee this, we simply created a new type of Condor VM in the Bologna Batch System, an *ALICE VM*, which would only run ALICE Jobs.

As before, we defined an expression to represent the third type of BBS VM, `IsAliceVM`. All ALICE machines are multi-VM machines, so the expression is based on how many of each VM type we wanted, like so:

```

NUM_SHORT_RUNNING_VMS = 2
IsShortRunningVM = (VirtualMachineID <= $(NUM_SHORT_RUNNING_VMS))
IsAliceVM = (VirtualMachineID > $(NUM_SHORT_RUNNING_VMS))

```

Next, we ensured that the ALICE VM would only run ALICE Jobs by defining a new START expression for its type:

```

IsAliceBBJob = ( $(IsBBJob) && TARGET.app =?= "ALICE" )
ALICE_VM_START = $(IsAliceBBJob)

```

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) ) \
          || ( $(IsAliceVM) && $(ALICE_VM_START) ) )

```

Next, we implemented policy requirement #, “ALICE BB Servers will not start long-running BB Jobs, but will start short-running BB Jobs when no ALICE jobs are running.”

By defining only ALICE VMs and short-running BB job VMs on ALICE BB Servers, omitting any long-running BB job VMs, we have already prevented long-running BB Jobs from starting – but to accomplish the second part of this requirement, we did the following:

First we created a NumVMsBusy shortcut, to keep track of whether or not the machine is already running as many (or more) jobs as it has CPUs⁵:

```

NumVMsBusy = ( ( (vm1_State=?="Claimed") + (vm2_State=?="Claimed") + \
                 (vm3_State=?="Claimed") + (vm4_State=?="Claimed") ) \
               - (State =?= "Claimed") )

```

Then we modified each ALICE BB server’s SHORT_RUNNING_VM_START expression as follows:

```

SHORT_BB_START = ((RemoteWallClockTime < 60*60) != False)

```

```

NON_BB_START = ( (LoadAvg - CondorLoadAvg) < 0.3 \
                 && LoadAvg < 0.5 \
                 && KeyboardIdle > (15 * 60) \
                 && TotalCondorLoadAvg < 0.5 )

```

```

SHORT_RUNNING_VM_START =
  ( ( ( $(IsShortBBJob) && $(SHORT_BB_START) ) || \
      ( $(IsNotBBJob) && $(NON_BB_START) ) ) \
    && ( $(NumVMsBusy) < $(NUM_SHORT_RUNNING_VMS) ) )

```

This preserves the previous short-running VM policy, but in addition prevents a short-running BB Job from starting unless the machine has idle CPUs. Since ALICE Jobs will always start, this indirectly ensures that short-running jobs are not started when ALICE Jobs are already running on all CPUs.

Next, we implemented policy requirement #, “Short-running BB Jobs already running on an ALICE BB Server when an ALICE Job arrives will be allowed to run to completion, at a higher (worse) OS priority than the ALICE Job.”

⁵ Note that we remove the current VM’s state from the count in NumVMsBusy. This is because the START expression (in which we will later reference NumVMsBusy) is evaluated not only by the negotiator at matchmaking time, but also by the startd itself after a match has been made, to double-check whether the job can still be started. If we included the current VM state in NumVMsBusy, it would allow an unclaimed VM to be matched, only to reject the job when it arrives, because its own state would have changed to claimed, altering the NumVMsBusy count.

This requirement was already met, since BBS jobs are never preempted. The only change needed was to adjust the default BBS priority expression as follows:

```
JOB_RENICE_INCREMENT = ( ( $(IsAliceBBJob) != True ) * 10 )
```

Finally, we added a convenient `bbs_submit_alice` submission script for users to use to submit ALICE Jobs. It is identical to the `bbs_submit_long` script, except for one additional line:

```
_CONDOR_APPEND_REQ_VANILLA='(BolognaBatchServer == True)'  
export _CONDOR_APPEND_REQ_VANILLA  
condor_submit -a '+BolognaBatchJob = True' \  
              -a '+LongRunningJob = True' \  
              -a '+app = "ALICE"' \  
              -a 'should_transfer_files = IF_NEEDED' \  
              -a 'when_to_transfer_output = ON_EXIT' \  
              -a 'universe = vanilla' \  
              $*
```