Chapter 9

CONDOR AND PREEMPTIVE RESUME SCHEDULING

Alain Roy and Miron Livny
Department of Computer Science, University of Wisconsin-Madison

Abstract Condor is a batch job system that, unlike many other scheduling systems, allows users to access both dedicated computers and computers that are not always available, perhaps because they are used as desktop computers or are not under local control. This introduces a number of problems, some of which are solved by Condor's preemptive resume scheduling, which is the focus of this paper. Preemptive resume scheduling allows jobs to be interrupted while running, and then restarted later. Condor uses preemption in several ways in order to implement the policies supplied by users, computer owners, and system administrators.

1. INTRODUCTION

Condor is a batch job system that allows users to take advantage of both dedicated and non-dedicated computers. Traditionally, batch job systems have allowed users to submit jobs only to dedicated local computers, such as multiprocessor supercomputers and clusters of computers. In addition to such dedicated computers, Condor allows users to submit jobs to computers that are only occasionally available for Condor to access, such as desktop computers belonging to other users or distant computers under someone else's control. We see such computers not as a problem, but as an opportunity, so we call them opportunistic resources.

The ability to use these non-dedicated computers provides users with extra computing power, but also adds complexity to Condor in two ways. The first is the need to remove jobs, called *preemption*, before they are have completed executing in order to meet the needs of owners, users, and administrators and to deal with unplanned outages. The second is the need to deal with the inevitable heterogeneity of computers available to Condor.

1.1 Preemption

In our experience [LL90], computer owners will only allow their computers to run Condor jobs if Condor does not negatively impact their activities. Therefore, Condor will checkpoint and preempt jobs when an owner needs their computer. This could either be when an owner returns to a computer after an absence, or when the computer is busy with some other activity, or for another reason based on the owner's policies. When another computer is available to run the job, Condor will resume the job on that computer. Note that the job can be resumed without loss of work because checkpointing saves the state of the job. This is *preemptive resume scheduling*, and the focus of this paper.

In order for preemption to be most useful, Condor needs to be able to checkpoint jobs so that work is not lost when the jobs are preempted. Condor can checkpoint most jobs without requiring the jobs to be modified if they can be relinked with libraries provided by Condor [LS92]. There are some restrictions on jobs that can use Condor's checkpointing, particularly jobs that spawn new processes or use kernel threads. In practice, many jobs can be relinked to use Condor's checkpointing, and thereby gain the benefits of preemptive resume scheduling.

Preemptive resume scheduling has been instrumental in Condor's success. Jobs can be preempted in order to meet the needs of any of the three types of stakeholders in a Condor system: users who submit jobs, owners of computers, and system administrators. Condor can preempt jobs on the behalf of users when better resources become available. Condor can preempt jobs on the behalf of computer owners to ensure that the owner's policy on sharing the computers is met. Finally, Condor can preempt jobs to meet the policy of the system administrators, who are concerned about the efficiency of the entire Condor pool of computers. In the remainder of this paper, we will discuss how Condor uses preemption in order to meet the goals of these three types of stakeholders.

1.2 Heterogeneity

While it is relatively easy to build a cluster of dedicated computers using identical or very similar types of computers, it is generally not possible to build homogeneous Condor pools which include non-dedicated computers. Computers within a department are often bought at different times and for different purposes, and it is common for them to be a variety of architectures and to have a variety of characteristics. Condor needs to be able to cope with this variety in order to provide the maximum amount of computational power to its users.

In order to deal effectively with this heterogeneity, Condor uses *matchmaking* to match user's jobs with appropriate computers. Both jobs and computers are described with the *ClassAd* (short for classified advertisements) language.

ClassAds provide schema-free descriptions of jobs and resources that is easy to use effectively, yet powerful enough to allow for sophisticated policies and constraints to be expressed by users, owners, and administrators. ClassAds and matchmaking are described in Section 3.

2. THE ADVANTAGES OF PREEMPTIVE RESUME SCHEDULING

Preemptive resume scheduling has several important properties.

- Preemptive resume scheduling allows the scheduler to take advantage of resources that may only be available occasionally. Because jobs can be checkpointed, preempted, and run elsewhere, work done on such a non-dedicated computer is not lost. Instead, this computer has provided an opportunity to accomplish extra work.

- Preemptive resume scheduling relieves the need to do backfilling, which is commonly done in schedulers. Backfilling allows a scheduler to take advantage of holes in the schedule to run more jobs and thereby increase its efficiency. With a preemptive resume scheduler, backfilling becomes much simpler, or even unimportant. It is not critical to schedule certain tasks during specific time-slices. Instead, any task can fill a time-slice, and it can be preempted later. Note that Condor does not use backfilling at all, since it does not expect users to specify time limits for their jobs, nor does it enforce any time limits.

- Preemptive resume scheduling helps to fairly share computers between users. For instance, as we show below, jobs can be preempted when a user that has a higher priority needs access to a computer. When combined with a system that provides dynamic calculation of user priorities, this can ensure that users are treated fairly.

- Preemptive resume scheduling allows high priority jobs to run when a high-priority user demands it by suspending lower priority jobs temporarily while the high priority jobs run. Because of the ability to resume preempted jobs, they can be easily resumed when the high priority job finishes. This is referred to as *computing on demand*.

3. SCHEDULING IN CONDOR

3.1 The Triumvirate

When users submit jobs to Condor, they do not submit to global queues, as they would in many other batch systems [BHKL00]. Instead, Condor has a decentralized model where users submit to a local queue on their computer,

and Condor processes on that computer interact with the Condor matchmaker and the computers that run the job. Interaction with the matchmaker is called matchmaking, and interaction with other computers is called claiming. Both of these interactions are essential steps to run a job.

Note that each computer in a Condor pool runs only a single job at a time, not multiple jobs. Computers with multiple CPUs may run one job per CPU.

Three distinct entities are involved in running a job: the owner of the job, the owner of the computer or computers that run the job, and the administrator of the entire Condor pool. Because each of these entities may have complex policies about running jobs, we can consider them a sort of triumvirate that controls how jobs are run. (The word "triumvirate" comes from the groups of three people that used to rule the Roman empire, such as Caesar, Pompey, and Crassus.) These individual triumvirates arise and fall as jobs are submitted and run.

To understand how matchmaking and claiming work, it is necessary to understand what ClassAds are. Please refer to Chapter 17 or [RLS98] for more information about ClassAds. From those descriptions, recall two important features of ClassAds: ClassAds are schema-free associations between names and expressions. Expressions can be evaluated in the context of another ClassAd, and this is the basis of matchmaking. For instance, a job ClassAd that contains an expression:

```
Requirements =    other.Type == "Machine"
               && other.RAM  >  500
```

says that the job must be matched with something that is a machine with at least 500 units of RAM. ClassAds do not define the units.

3.2 Matchmaking and Claiming

In order to run a job submitted to Condor, there are interactions between three components, as shown in Figure 9.1: the *user agent*, the *owner agent*, and the *matchmaker*. These components represent the triumvirate and ensure that their wishes are followed.

Note that in other Condor literature, you may find the user agent referred to as the *schedd*, the owner agent referred to as the *startd*, and the matchmaker referred to as two programs, the *collector* and the *negotiator*. These are not obvious names, but they persist for historical reasons.

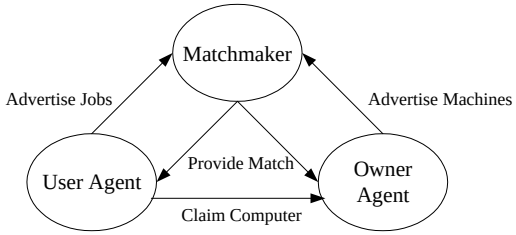

Figure 9.1. The Condor triumvirate which governs where jobs are run and when they are preempted.

3.2.1 Participants in Matchmaking and Claiming

There are three components to the triumvirate, and they are responsible for representing the needs of the people or groups they represent.

- *User Agent* When a user submits a job, it is the user agent's responsibility to make sure that the job runs to completion, assuming no errors in the user's code. It maintains a persistent queue of jobs and a history of past jobs. If other components in Condor fail, it is responsible for retrying the job and informing users of problems that occur.
- *Owner Agent* When a owner decides to add a computer to the Condor pool, the owner agent ensures the owner's policy for how the computer can be used is enforced. It is also responsible for starting jobs that are submitted to the computer.
- *Matchmaker* The matchmaker is responsible for finding matches between user and owner agents, and is also responsible for implementing pool-wide policies that affect the overall performance and stability of the entire pool.

3.2.2 The Process of Matchmaking and Claiming

When a user submits a job to the user agent, it is immediately stored in a persistent queue. This persistent queue allows the user agent to recover should the agent or the computer on which it is running fail. The job is uniquely identified by the name of the user agent, which is unique within the pool, plus a number unique to the user agent. The user agent then sends a ClassAd that informs the matchmaker that it has jobs to run. As long there are jobs that are not running, the ClassAd is sent to the matchmaker every five minutes. The user agent does not rely on stability or persistence in the matchmaker.

Similarly, every five minutes each owner agent in a Condor pool submits a ClassAd that describes the computer it is responsible for. ClassAds are also

sent whenever the state of a computer changes significantly. For instance, the state of a computer can be "idle", meaning that it is available to run jobs, or "owner", meaning that the owner of the computer is using it; when the state changes from idle to owner or vice-versa, the owner agent will inform the matchmaker.

The matchmaker accepts ClassAds from user and owner agents, but does not store them persistently. Instead, the matchmaker discards ClassAds if the job or computer ClassAds are not resubmitted for a while—they are presumed to be unavailable for running jobs if they do not regularly report themselves as available. Should the matchmaker ever crash, it will simply relearn the state of the Condor pool within five minutes. This soft-state mechanism enables reliable operation without complicated persistence mechanisms.

Whenever a job is submitted, or every five minutes, the matchmaker attempts to find matches between jobs and computers. This is called the negotiation cycle. It contacts each user agent that has jobs to run, and obtains the ClassAds that represent the job. It then matches the jobs against each machine ClassAd in the pool to see if it can run the jobs. In order to match, each job's requirements (as given in the job's ClassAd) are evaluated in the context of the machine and the machine's requirements are evaluated in the context of the job. These both must evaluate to true in order for the job to match.

When a match is found, the matchmaker informs both the user agent and the owner agent of the match, then continues matchmaking. It is up to the user and owner agents to claim the match independently of the matchmaker. To do this, the user agent contacts the owner agent. Because the matchmaker was operating on information that may have been out of date, the owner checks to make sure that the job and machine ClassAds still match. This is useful because, for instance, the machine's owner may have reclaimed the machine for his own use since the matchmaker performed that match, and therefore no jobs are able to run on the machine.

If the match can still be made, then the user agent sends the job directly to the owner agent and it begins executing. The user agent monitors the progress of the job, and should any problems outside of the program scope [TL02] occur, such as failure of the owner agent, the user agent will go through the matchmaking process again. If the job fails on its own accord, such as a segmentation fault due to a pointer problem, the user agent records the error and informs the user.

When a job begins running on a computer, a new process on the same computer as the user agent begins running. This process is known as a *shadow*, and it is responsible for implementing Condor's remote I/O capabilities. If a job has been relinked with the Condor libraries in order to be able to be checkpointed, as described above, the shadow will perform two functions. First, it will assist in checkpointing if necessary, either saving checkpoints directly on

the computer the job was submitted from or redirecting them to a specialized checkpoint server. Second, it will redirect I/O on behalf of the application. That is, all I/O performed by the remote application will be performed on the computer that the job was submitted from. This allows jobs to have access to their files no matter where they execute, and also makes them less dependent on free disk space from the remote computer.

Note that running jobs continue even if the matchmaker fails for any reason because all communication is between the computers that submitted the jobs and the computers that are running the jobs. While the matchmaker is unavailable no new jobs can be matched, but all old jobs continue on with no difficulty. When the matchmaker becomes available, jobs will continue running within five minutes.

This is just a sketch of the matchmaking process, and there are several complicating factors. In particular, there are several ways in which jobs can be preempted while running. These are described below.

4. PREEMPTION IN THE TRIUMVIRATE

As described above, running a job in a Condor pool is a cooperative process between three agents that represent the concerns of three people, or groups of people. The user agent represents the person who wishes to run a job, the owner agent represents the person who owns a machine, and the matchmaker represents the maintainer of the Condor pool. The interests of all the people involved must be respected. In part this happens by allowing users and owners to specify requirements for a successful match. Another important aspect is preemption: each of these three parties is allowed to preempt a running job in order to satisfy their requirements.

In order for preemption to be the most effective, checkpointing should be used to allow the state of the program to be saved. When programs meet certain requirements (such as not using long-lived network communication) and can be relinked with Condor libraries, they can be checkpointed so that they can be restarted after they are preempted. Condor can checkpoint jobs when they are preempted from a machine, users can manually request that checkpoints be made, and Condor can periodically checkpoint jobs to ensure against future failures. Not all jobs can be checkpointed, so preemption must be used cautiously. But from a user's perspective, preemption is an inevitable fact of life: even in a dedicated cluster of computers, there may be crashes and disruptions that cause jobs to fail during execution. Note that in a Grid environment, this may happen even more frequently. Being able to checkpoint and react to preemption, for whatever reason it might occur, is an essential part of Condor's approach to reliability.

4.1 User Preemption

There are three ways that a user can preempt a job.

The first way is manually: a user can always remove a job that is either waiting to run or running by executing a command. The job can be checkpointed then rerun, or it can be forcefully terminated, which does not allow the job to be checkpointed and restarted from where it was terminated.

The second way is an automation of the above process. If the user knows some conditions under which the job should be removed—perhaps the user knows that when the job has run for more than 12 hours it must be doing something incorrectly—these conditions can be specified when the job is submitted. Condor's user agent will monitor this condition, and should it ever become true, it will stop the job on behalf of the user. As of Condor version 6.4, the job is not checkpointed, though this is likely to change in future versions.

The last, and most interesting way, is that Condor could periodically rematch the job in order to see if could run on a better machine. If a better machine would be found, Condor could preempt the job on the machine on which it is running, then restart it on the better machine. This is not currently implemented, but Condor is structured to make this easy to implement. Most of the changes would take place within the owner agent which would continue to advertise a job while it was running, but would change the Requirements expression in the ClassAd to add the requirement that "Rank > CurrentRank" to state that it will only run better machines. The ease of this implementation shows the usefulness of the structure of Condor—the owner agent can easily implement a user's policies for jobs. We expect this feature to be implemented in the near future.

Note that, although adding the mechanisms to do this is not difficult, it needs to be done carefully in order to avoid problems. For instance, it is important to ensure that jobs do not "thrash" between computers, always looking for greener grass elsewhere. To do this, a job's requirements either need to specify features that are not affected by running on a machine, such as total memory instead of system load, or a job's requirements need to be updated to restrict a job from running on machines it has already run on.

4.2 Owner Preemption

If an owner wants to remove a Condor job running on his machine, there are two ways for it to happen.

First, Condor can automatically preempt jobs. Owners can write flexible policies that state when jobs should be preempted. For example, an owner could write a policy that when he begins typing at the keyboard, jobs with a small image size are suspended and other jobs are preempted. (Condor calculates the image size for each job and puts it into the job ClassAd, so this

information is available.) If a job is suspended for more than ten minutes (the owner has continued typing on the keyboard), the job must be preempted. The owner can choose to allow or disallow checkpointing before the job is preempted. Of course, various alternative strategies can be used: all jobs could be suspended, all jobs could be preempted, other criteria can be used to decide when it should occur and so on. The owner gets to decide how Condor jobs are allowed to use his resource and Condor enforces his desires.

Second, a user can always request that Condor preempt jobs running on his machine by running a command. This is not usually necessary because of Condor's mechanisms for automatically preempting jobs.

4.3 The Matchmaker: Preemption for Everyone

When the matchmaker is looking for locations to run a job, it can enforce the desires of the pool administrator by attempting to place jobs in order to increase the efficiency of the pool or to favor some users instead of others. The matchmaker also helps owners run jobs that they prefer by preempting other jobs already running. To understand how both of these work, we will describe the matchmaking process in a bit more detail than we described it above.

When the matchmaker is looking for a match for a particular job, it examines the ClassAd for each available computer one at a time, in order. Note, however, that computers continue to advertise themselves while they are already running a job, because they are willing to run "better jobs". This advertising allows currently running jobs to be preempted.

After checking that a particular job and machine match, the matchmaker evaluates several expressions:

- How does the job rank the machine? Job ClassAds specify a rank expression that, when evaluated, can differentiate between different machines. For instance, a job might prefer a faster machine, a machine with more memory, or a machine belonging to the research group that the job's owner is part of. This machine rank allows Condor to meet the user's needs.

- Is there already a job running on the machine? If so, the matchmaker evaluates the machine's rank of the job and compares this rank to the machine's rank of the currently running job. If the new rank is higher, the matchmaker may consider preempting the job. The relies on a rank expression supplied by the machine's owner in the machine's ClassAd to describe which jobs the owner prefers to be run on his machine. This job rank allows Condor to meet the owner's needs.

- If there is a job running on the machine, but it does not have a higher rank than the currently running job, does the owner of the job looking

for a match have a higher user priority than the owner of the currently running job? If so, the matchmaker evaluates the pool administrator's requirements for preempting to see if they are true. The pool administrator may only allow preemption based on user priority when certain conditions are met, such as not preempting jobs when they have not run for very long, in order to avoid thrashing. This allows Condor to meet the system administrator's needs.

This machine will be a candidate for running a job if:

- the job's rank of the machine is better than any seen so far,
- the job's rank is the same as the previous candidate, but running the job on this machine would do "less" preemption (either no preemption, or preemption based on rank instead of preemption based on priority), or
- the rank is the same as the previous candidate, and the same kind of preemption would be performed, but the pool administrator ranks this as a better preemption than the previous candidate. The pool administrator may rank a job preemption as better for any reason, but common reasons might be that a job is smaller (so there will be less network traffic caused by the preemption) or that the job is being run by a more important user.

If this machine is the best candidate so far, it is used for future comparison with machines as matching for a job continues.

As described above, this matchmaking algorithm helps enforce the policies of the pool administrators, the owners of machines, and the users who submit jobs. For owners of machines, if a machine is running a job but finds another job that is better (perhaps it is a job submitted by a collaborator of the machine owner), it can choose to run the new job. Also note that Condor will strongly prefer not to preempt jobs at all, if it can run a job on an idle machine. For pool administrators, Condor lets the administrator impose policies on when jobs can be preempted, and which jobs should be preferentially preempted over other jobs.

5. CONCLUSIONS

By using all of these different types of preemption together, Condor can balance the desires of the users, machine owners, and pool administrators. Preemption is supported throughout Condor by providing both checkpointing mechanisms and opportunities for preemption. Because of this support, Condor can both take advantage of sporadically available resources such as desktop computers and can react to problems such as machines that fail while jobs are running. This flexibility and robustness has been essential to Condor's success.