

BOINC and Condor – Scavenging the Scavenger

Derek Wright, wright@cs.wisc.edu, UW-Madison Condor Project (www.condorproject.org)

What is Condor?

Condor is a system for managing *high-throughput* computing -- huge numbers of tasks over long periods of time, not short bursts of very fast computation. Condor was started in the early 1980's primarily as a way to scavenge and harness wasted computing cycles on idle desktop workstations. It has evolved over the years into a system for managing all sorts of computing resources (workstations, "big iron" SMP machines, dedicated clusters, computational grids, etc), data movement, authentication, and more. In many ways, Condor could be seen as an ancestor of BOINC, and they share some common roots at the University of Wisconsin, Madison.

One of the novel features of the Condor system is its generic scheduling based on *match-making*. Every entity in the Condor system advertises itself with a *classified ad*, much like the ones in the newspaper. Each machine describes itself, what it has to offer, and what it is looking for. Similarly, jobs describe themselves and what kinds of machines they are looking for. Each party can define requirements that must be met, and can rank their preferences across all entities that meet their requirements.

Resource owners define their own policy for when jobs should run, if they should be suspended or evicted, etc. The machine's requirements are a central part of this policy – what kinds of jobs should be allowed to run under what conditions.

One part of the Condor system, the *negotiator*, periodically goes through all the requests and finds machines that match jobs (both sides have to want each other). Because the attributes in the classified ads are dynamic and customizable, as are the policy expressions themselves, this system provides enormous flexibility in scheduling jobs and resources.

Can BOINC and Condor work together?

One of the problems that Condor faces is that there can be periods when there are no jobs in the system. This results in idle (and therefore wasted) resources. One of BOINC's many strengths is that by participating in multiple projects, there's always work to be done. To take advantage of this never-ending supply of useful work, we modified Condor to have knowledge of the BOINC client. *Administrators can now configure machines such that if there are no Condor jobs available, Condor will hand control over the resource to BOINC.*

How does this integration work?

The Condor daemon that manages compute nodes is the *startd*. The *startd* enforces the policy for when jobs should run and controls their execution. Each compute slot managed by a *startd* has a state, and the machine's policy expressions control when certain state transitions happen.

When the *startd* has an idle compute slot, it evaluates an expression that controls if the slot should enter a *backfill* state. Upon entering this state, the *startd* spawns the command-line BOINC client (as defined in the Condor configuration file). This client is the standard, un-modified BOINC client, and is configured to join whatever computing projects the machine owner wants. Effectively, the Condor *startd* serves the same purpose as the BOINC screen-saver: it decides when the resource is otherwise idle, and tells BOINC it should put it to use.

While a Condor resource is in the backfill state, if an interactive user comes back to the machine, or if a Condor job is matched with it, the *startd* will kill the BOINC client and any processes it has spawned.

In this way, we are now using BOINC to scavenge unused cycles from Condor, itself a scavenger of wasted compute power.

Possible Areas of Future Work

Dynamically sharing compute slots

With dual-core CPUs becoming the norm, multiple compute slots per machine are the wave of the future. Currently, when either BOINC or Condor are running, unless specifically told in advance, they both think they should discover all physical compute slots on the machine and control everything. Ideally, Condor would be able to add and remove resources (including RAM) from BOINC's control, so that instead of killing the BOINC client entirely (all-or-nothing) they could more peacefully co-exist.

Identifying BOINC tasks within Condor

If a BOINC work-unit running on a compute slot is about to miss a deadline, and there's another resource working on a work-unit that has a lot of time before it expires, Condor should prefer to preempt the task that's not going to expire, and allow the one under a tight schedule to complete (if possible). The challenge here is getting data about the expiration date of the current work unit out of the BOINC client and giving it to the startd so that this information can be included in the eviction policy expressions. Condor provides a hook for dynamically adding attributes into the machine classified ads, so this task might involve no changes to the Condor source, and only minor changes to BOINC's code.

Identifying Condor resources in BOINC

Due to the fact that Condor compute slots are often more tightly administered and managed than random screensavers on the Internet, these resources could be identified to the BOINC servers as such. This would allow BOINC to more effectively schedule work units as a given computation is nearing completion. A huge computation that is 98% done can be held up for many days, waiting for the last few work units to return their results. By identifying more reliable Condor slots within the BOINC scheduling algorithm, we could shorten these periods of "draining" the jobs.

Condor's Master-Worker (MW) system

MW is a system built on top of Condor for managing a pool of work. There is a C++ API for defining work tasks, and a master that knows how to claim Condor resources, create a pool of slots, and allocates the work. MW attempts to ensure it has enough slots at all times, will re-try a task if it hasn't gotten an answer back, and takes care of most of the complication for application writers facing similar problems. In these respects, MW is *very* similar to BOINC.

BOINC and MW could be modified to talk directly to each other, to the benefit of each:

- * MW could hand tasks (aka work units) over to BOINC, to gain access to a much wider (though potentially less reliable) pool of resources.

- * BOINC could hand work units to MW for an even more guaranteed and reliable way to drain a computation as it nears completion. Instead of the final work units that are blocking a whole computation running on Condor nodes as backfill jobs, these critical work units could run as first class Condor jobs via MW.

Better packaging and installation

Currently, BOINC must be downloaded, installed, and configured separately from Condor. We'd like to ship a pre-configured copy of the BOINC client directly inside Condor release packages.

Collaborations on portability issues

- * We currently run nightly builds of the latest development version of BOINC on our automated build and test infrastructure (<http://nmi.cs.wisc.edu>), to verify BOINC's portability on over 20 different platforms.

- * The BOINC client and the Condor startd both have to discover similar attributes of the machines they run on (RAM, # of CPUs, keyboard activity, etc). Condor has a library to abstract these details and provide a single portability layer. Perhaps we could share code and work together to maintain it.