Education

Teaching Condor Grid Computing to Beginning Programming Students

Amy Apon and Matt Baker • University of Arkansas

We have been teaching grid computing at the University of Arkansas since 2004, focusing on the development of grid services, the Globus Toolkit, and higher-level grid components. An early course, Cluster and Grid Computing, successfully taught Globus materials to advanced undergraduates and beginning graduate students (See http://www.csce.uark.edu/~aapon/courses/gridcomputing/index.html for a description of the course, which used Globus Toolkit 3). However, the material assumed advanced knowledge of Java programming and covered networking and basic distributed systems concepts, and was therefore too advanced for beginning programming students. If we wanted to teach grid computing to beginners, we would need to focus on creating and submitting a grid job rather than programming grid services. We also wanted to teach grid computing to advanced science and engineering students whose programming skills were similar to those of beginning computer science students.

Beginning with our involvement in the development of a regional middleware test bed (http://archie.csce.uark.edu/gpn), we helped create the GPNGrid virtual organization (http://collaboration.greatplains.net/wiki/index.php/Main_Page) within the Open Science Grid (http://www.opensciencegrid.org). The OSG uses grid-enabling software called the Virtual Data Toolkit (a packaging and distribution of Condor and Globus). Condor handles workload management, while the Globus software manages security, transport, and other services in the grid. Condor is also a well-established tool for high-throughput computing (http://www.cs.wisc.edu/condor) separate from a Globus-enabled grid. Because of our positive experiences with OSG and Condor, we decided to teach Condor to beginning students as a grid computing tool.

Condor basics

We run a small cluster of computers on the OSG. It is set up so that users can submit jobs to either the local Condor pool or via a user certificate that is registered with the GPNGrid virtual organization. With the first approach, a student needs only an account on the local grid machine and an understanding of the basic Condor command-line interface. Although installing and maintaining the complete OSG stack requires significant commitment, the Condor grid was initially installed by a graduate student—so, this task is a feasible semester-long graduate student project.

Condor is based on the concept of a classified advertisement, or ClassAd. You can use a ClassAd to advertise machines with specific available resources (such as one or more processors or a certain amount of memory) or to describe the resource requirements of a job you need filled. Condor acts as a matchmaker between machines and jobs. Once the job starts executing on the resource, Condor tracks it. When the job completes, Condor writes the output to standard out to the user-specified file.

The ClassAd mechanism is powerful enough to specify the execution of multiple copies of a job with different parameters and different output or input files. Figure 1 shows a simple Condor ClassAd. The executable is called fire. The single argument to the fire program is the Condor process ID. ClassAd will execute five copies of the program (as specified by the line "queue 5"), each with a single argument (the number 0, 1, 2, 3, or 4). The output to standard out of process 0 will be written
to the output file, "fire_0.out," and the output to standard error will be written to the file, "fire_0.error."

```
universe = vanilla
executable = fire
arguments = $(PROCESS)
output = fire_$(PROCESS).out
error = fire_$(PROCESS).error
log = fire.log
queue 5
```

Figure 1. A Condor ClassAd that submits five copies of the fire program.

Condor offers many features, including check pointing of interrupted jobs, job migration, execution of parallel and Java jobs, and workflow management using a directed acyclic graph manager.

**First attempt: Fall 2005**

Our first experience teaching Condor grid computing to beginning students was at the end of the Programming Foundations I class of fall 2005. All freshman computer science and computer engineering students take this course. Typically between 30 and 50 percent of the students are advanced undergraduate students from other majors (primarily engineering majors that require a programming course). The course is taught in C. Programming topics include data types, control structures, functions, arrays, files, C structs, and the mechanics of running, testing, and debugging. The course includes a weekly two-hour laboratory session that covers Unix skills and practical programming exercises. The only prerequisite was a college-level algebra course (which we have since changed to calculus.)

We introduced Condor to freshmen as a one-hour lab exercise (http://comp.uark.edu/~aapon/condor/CondorWithFiles/grid.html) in the last lab of the semester. Before the lab, we presented a one-hour lecture on basic Condor concepts (including writing a ClassAd), and the students completed a programming assignment (http://www.csce.uark.edu/~aapon/courses/foundationsIfall2005/homework/hw7.html) where they wrote a simulation of the Game of Life by Conway. For the programming assignment, we gave students substantial starting code and then asked them to complete the implementation, which required manipulating a two-dimensional array.

The Game of Life uses a starting configuration—in this case, from an input file—and then runs for a fixed number of generations. When the simulation ended, students could print an output file that showed the final configuration. The scientific question we posed was, "Given a set of input configuration files, which of these will still have living cells after 20 generations of the simulation?"

We asked students to use Condor to submit a set of jobs that would each read a unique input file and then produce a unique output file. We modified the program to print not just the final configuration but also whether it had any living cells. Students then hand-tabulated which input files persisted for at least 20 generations.

This assignment was partially successful. Students could submit Condor, especially in the lab setting, and could read and tabulate the results. However, several concepts were difficult and required much more explanation than we had anticipated:

- **The batch-submission process.** This was surprising to someone who learned to program on a mainframe! However, all the programming prior to this lab had been interactive, using command-line Unix tools. The concept of submitting a job, having the results appear in a file "automatically," and retrieving those results at a later time required a conceptual adjustment for some students.
Using the computer to solve a scientific problem. Before this exercise, programs had focused on learning programming techniques. One homework assignment was to write an interactive text-based computer game, and several involved writing reports. However, using the computer to answer a scientific question by searching a set of parameters using a program that runs several times on different input data was a new concept.

The execution’s distributed nature. Students could type `condor_submit myClassAd` and see several output files appear in their directory, but it was not obvious that the execution was happening on several of 30 computers set up in the Condor pool.

Another problem occurred when the Condor pool failed because of physical limitations on the computer that was running our Condor manager. That computer had just 512 Mbytes of memory. The Condor manager maintains a record in memory for each Condor job submission. When 25 students in the lab submitted jobs at the same time, they overloaded the manager, which meant that jobs were not logged and completed. Additionally, the error message was confusing to the students. Because this was the last lab of the semester and students could drop one lab grade, many never finished the exercise.

Overall, students thought that Condor was cool, but they needed substantial guidance to complete the exercise, and it was not clear that they had the necessary understanding to complete another Condor exercise on their own.

Second attempt: Spring 2006

Despite these difficulties, we modified the exercise and attempted to teach Condor again in the same course the following semester. The fall class had more than 125 students in a large lecture hall, but the spring class (the off semester) had only 35 students. The same graduate assistant taught the two lab sections. We hoped that the smaller, more intimate setting would make teaching Condor easier. Fewer students and a more robust computer for the Condor manager also helped us avoid another system failure.

We developed a fire simulation program (http://comp.uark.edu/~aapon/condor/BasicCondor/fireexercise.html) that did not require input files. As before, we introduced students to a program that performs a simulation on a two-dimensional grid. This time, we covered the code in lecture but did not have students write or modify it. We gave them a Condor ClassAd and instructions on submitting and running Condor applications. As before, we gave a one-hour lecture on Condor basics. Lecture material also covered batch submission and computational science. Because a portion of the course teaches Unix utilities, the lecture also covered the Unix `cat` and `sort` tools to process the resulting output files from Condor. The scientific question posed was, “What percentage of the forest will burn with a given probability of a neighbor tree catching fire?” In a lab setting, we asked students to run the Condor application and then graph the results. Because they did not know Unix utilities for graphing, students copied the data file to their Windows desktop and constructed the graph using Excel.

The results still were not entirely satisfactory. Again, students could complete the mechanics of submitting a Condor application, retrieve the results, and construct the graph. However, they did not understand the distributed grid computation. The Condor submission process seemed to prevent students from understanding the computational science concepts and using the computer to answer a scientific question. Also, it was cumbersome for them to use Windows tools to construct the graph while completing the computation on the Linux server.

Third attempt: Fall 2006 and spring 2007

During fall 2006, rather than teach Condor directly, we wrote a homework assignment to teach students computational science concepts and how to use the computer to answer a scientific question. The fall class was large, with almost 150 students and seven lab sections. About 65 percent of the
students were computer science or computer engineering majors taking the first major course, and most of the remaining students were juniors or seniors from other engineering majors.

The homework assignment used a 2D array to simulate the heat distribution (http://comp.uark.edu/~aapon/condor/ComputationalScienceIntro/heatdist.html) in a region. It used graphical output on the Linux system directly. Figure 2 illustrates student output for one program run. Students also wrote a short report analyzing the result of running the program with different input data.

Figure 2. Graphical result of the heat distribution problem; area 30 × 30, heat source (3, 3).

This assignment was very successful, with nearly all students completing the assignment and the report. When we taught computational science concepts separately from batch submission and distributed computing, students could use the computer to answer a scientific question.

During the spring 2007 semester, we offered a special studies course that covered Condor and grid computing as a follow-on from the fall Programming Foundations I course. As before, lecture material covered the basic Condor and batch submission concepts. It also covered utilities for processing the output files from Condor, including cat and sort, and the addition of simple graphing using Linux gnuplot. The first assignment was to complete the fire exercise from spring 2006. The task (a one-week homework exercise) was to write a program to calculate the averages of several results, write a Condor ClassAd that would submit 101 jobs for calculating averages over a range of 0 to 100, and modify a gnuplot script to complete the graph for all results. This assignment was successful, and it indicates that a first exercise emphasizing computational skills was adequate preparation for the next steps of using Condor and Unix utilities in grid computing. Figure 3 shows a student-created output. The graph contains the results of over 10,000 simulation runs.
Figure 3. Student output of Condor simulations graphed using gnuplot.

Although the special studies course was a small sample set, our approach—beginning with a first assignment on computational concepts, then adding concepts of batch submission and grid computing incrementally—appears to be successful for teaching Condor grid computing to beginning students. (Students might also consult additional resources, such as beginning textbooks—for example, *Introduction to Computational Science*, Angela Shiflet and George Shiflet, Princeton Univ. Press, 2006. The Game of Life and Fire simulations, and many other computational science materials, are available from the Shodor Education Foundation at http://www.shodor.org.) We are now structuring and extending our exercises into a series of assignments to illustrate different grid computing features using the Condor workload manager—including the use of input data files and Condor Dagman, acquiring a user certificate, and running applications on the OSG. Possible applications could include incorporating Condor grid computing as a unit of a computational science course or a second programming course.

**Acknowledgments**

This work is supported in part by National Science Foundation grant DUE #0410966 to teach grid computing to undergraduate computer science students.

**Amy Apon** is an associate professor at the University of Arkansas. Contact her at aapon@uark.edu.

**Matt Baker** is a PhD student and graduate assistant at the University of Arkansas. Contact him at mmb05@uark.edu.

**Related Links**


Cite this article: