Co-Scheduling Data and Computation using Condor and SRM

A collaboration by UW-Madison and Lawrence Berkeley National Labs
Presented by Derek Wright, Paradyn/Condor Week 2005, Madison, WI

What problem are we solving?
Many computational problems require access to extremely large (and rapidly growing) datasets. This data is often housed in mass storage tape libraries, where the cost of retrieval is high. During analysis, users typically extract subsets of the data and perform computations on them. It is a well-known difficult problem to build systems that execute the computations and data movement in a coordinated manner. Our solution is an attempt to create such a system using existing mature technologies: Condor and SRM.

What is Condor?
What a silly question at a conference like this. You haven't been paying close enough attention. Read www.cs.wisc.edu/condor if you must.

What is SRM?
SRM stands for "Storage Resource Manager". It is a system developed at Lawrence Berkeley Labs under the guidance of Arie Shoshani to manage and provide a consistent interface to many types of data storage devices. An SRM can manage free space, file lifetimes, file sharing, caching policies, garbage collection, access to remote file locations, and other functionality. There are different kinds of SRMs that interface with various types of storage. For example, there are DRMs ("Disk Resource Managers") to manage disks, TRMs ("Tape Resouce Managers") to manage tertiary storage systems, and HRMs ("Heirarchal Resource Manager") to manage staging files from tape library to a disk cache (a combination of a DRM and a TRM).

How will these solve the problem?
Our solution to the co-scheduling problem is to start with a compute cluster where each node only has a local disk cache (no shared file system for the data). We use a DRM to manage the local disk cache on each node, and give Condor access to information about the files in the cache for scheduling the computational jobs. The DRM handles all of the problems of the disk cache itself: garbage collection, staging files, managing file lifetimes, file sharing, etc. Condor uses matchmaking to schedule jobs on compute nodes that have the data the job is looking for, and handles all the work to spawn, monitor, and manage the computational tasks. *The contribution of our work is to allow Condor to use knowledge of the state of the disk cache for matching computation with data.*

How does Condor talk to the DRM?
The Condor daemon that manages compute nodes is the startd, which generates a ClassAd to describe the machine. There is a hook in the startd for generating dynamic data that can be inserted into these ClassAds. This hook was added to support the Hawkeye monitoring tool, but is present in every startd as the "STARTD_CRON_JOBS" config file setting. Using this hook, we wrote a simple script to query the DRM for the files in the cache and publish those in the ClassAd for that node. However, since the number of files in the cache can be enormous, we only query for files that are requested by a job currently in the system and publish information about those files. This keeps the size of the ClassAds much smaller, and therefore, easier to use.

Did you have to modify Condor or SRM?
We didn't modify Condor at all. Everything in our system uses the stock Condor distribution off the web site. SRM was only modified to provide a query interface so that the startd's monitoring script could get a list of files in the DRM's file cache.

How does it work?
An end user submits an analysis task and a description of the necessary data subset to a component called a "Job Decomposition Module" or "JDM". The JDM breaks up the task into individual Condor jobs, each one with a requirement for a specific file in the data set. The JDM gives a list of files and data requests to the "File Scheduling Module", or "FSM".

The FSM maintains a list of the files required by all jobs in the system and how many jobs require each file (the file's "popularity"). The FSM provides this list of files to the script the startd

uses to query the DRMs. The FSM submits all tasks as individual Condor jobs monitors their progress by reading the UserLog written by Condor (just like DAGMan does). Each job has its own `requirements` expression to match a machine with the right file(s).

The startd publishes its usual machine ClassAd but also includes the information about files in the disk cache on each node. The regular Condor negotiation cycle matches idles jobs in the queue with available machine slots where a required data file already exists. If the FSM notices files that are needed by jobs but do not exist in the system, it schedules a file staging request from the HRM controlling the mass storage device. If a large number of jobs require the same file and that file is only on a small number of nodes, the FSM can also schedule the file to be replicated onto other compute nodes, either from the HRM or a neighboring DRM.

Who makes scheduling decisions (and how)? The main scheduling decisions are made by the FSM. Since the cost of moving files from tertiary storage is high, the important decisions are when and where to stage files. The trick is to find the right algorithms for staging files to optimize the behavior of the system in various dimensions. Do you want to minimize the total number of transfers from tape? Do you want to maximize the total utilization of the cluster? Do you want to minimize the aggregate time jobs wait in the queue? Currently, we are experimenting with different file staging algorithms and measuring these metrics. We are hoping to present our results at the SSDBM'05 conference.

One of the most promising algorithms so far involves maintaining the notion of the total file popularity of any disk cache. Here, we add the total popularities of all the active files in each cache. All file staging and replication decisions are made based on this total popularity. If a disk cache's popularity reaches 0, there are no jobs in the system that want any of its files, and we need to start staging new files there. When we decide to replicate a hot file, we find the machine with the lowest total popularity.