Pseudo-interactive monitoring in distributed computing

I Sfiligoi[1], D Bradley[2] and M Livny[2]

[1]Fermilab, Batavia, IL 60510, USA
[2]University of Wisconsin, Madison, WI 53707, USA

E-mail: sfiligoi@fnal.gov, dan@hep.wisc.edu, miron@cs.wisc.edu

Abstract. Distributed computing, and in particular Grid computing, enables physicists to use thousands of CPU days worth of computing every day, by submitting thousands of compute jobs. Unfortunately, a small fraction of such jobs regularly fail; the reasons vary from disk and network problems to bugs in the user code. A subset of these failures result in jobs being stuck for long periods of time. In order to debug such failures, interactive monitoring is highly desirable; users need to browse through the job log files and check the status of the running processes. Batch systems typically don't provide such services; at best, users get job logs at job termination, and even this may not be possible if the job is stuck in an infinite loop. In this paper we present a novel approach of using regular batch system capabilities of Condor to enable users to access the logs and processes of any running job. This does not provide true interactive access, so commands like vi are not viable, but it does allow operations like ls, cat, top, ps, lsof, netstat and dumping the stack of any process owned by the user; we call this pseudo-interactive monitoring. It is worth noting that the same method can be used to monitor Grid jobs in a glidein-based environment. We further believe that the same mechanism could be applied to many other batch systems.

1. Introduction

Many scientists require large amounts of compute power, well in excess of any workstation they can afford. For this reason, most of them resort to distributed computing to satisfy their needs. And handling tens, or hundreds, of compute resources by hand is usually not an option; so a workload management system, such as a batch system, is usually needed.

Most of the time, scientists like the batch system paradigm. All they need to do is split the problem into manageable pieces, define the dependencies, and submit the obtained workflow to the batch system. They can then concentrate on other topics while waiting for the notification that the results are ready; the batch system will transparently handle the users' workflows in the most efficient manner.

However, sometimes things don't go as expected; the computation may take longer than expected or the returned results may be corrupted or meaningless. In such cases, users need to be able to peek at the status of the workflows, going into as much detail as needed to find and fix the problem; i.e. they need interactive(-like) access to their running jobs to browse through the job log files and check the status of the running processes. Unfortunately, most batch systems are based on the black-box paradigm, making this exercise a very painful, if not completely impossible endeavor, as sketched in Fig. 1.

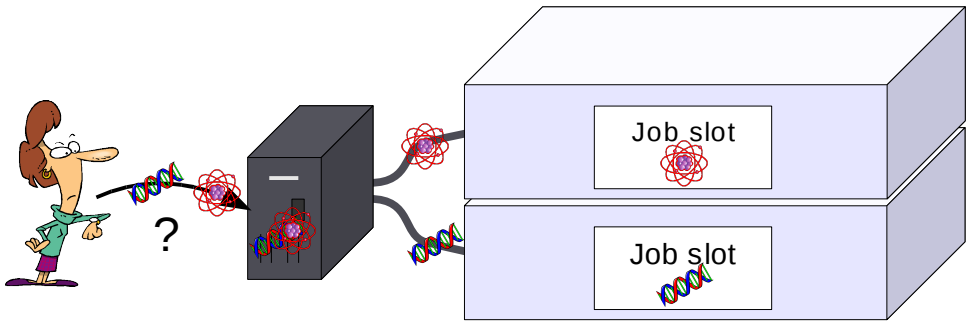

Figure 1: When jobs misbehave, users are often left wondering

2. Monitoring capabilities of popular batch systems

The monitoring capabilities of the currently popular batch systems vary a lot, but to our knowledge none provides true interactive-like monitoring capabilities out of the box. As an example, let's look at four use cases; PBS-like batch systems, dedicated Condor pools, direct Grid submissions, and Condor glide-in based use of Grid resources:

- PBS-like batch systems[1] (like OpenPBS and Torque/Maui) provide very limited monitoring. Users can know if and on what worker node a job is running, but no details about what the running job is doing. Some system administrators work around this limitation by giving users login access to the worker nodes, but this ability can easily lead to abuses; e.g. users running their jobs in interactive mode.
- Dedicated Condor pools[2] have similar limitations; users can know if and on what worker node the job is running, why is it not yet running, as well as what the system load is where the job is running. But there is no detailed information about what the job is doing. Giving users login privileges is again the most frequently deployed workaround, with the associated abuse risks.
- Grids[3] have become quite ubiquitous, and many sciences are relying on them to handle the needed compute resources. While the definitions of “Grid” are many, we will here consider a Grid as just a set of independent batch system clusters, each managed within its own administrative domain, but with a common outside batch interface and a common authentication model. An inter-cluster workload management system, like Condor-G or the gLite WMS, may be provided. The monitoring limitations of this environment are particularly severe; the best a user can hope for is knowing if a job is running and on what cluster. If more information is needed, the user needs to contact directly the managers of the remote batch system cluster, as the common interfaces typically don't support such functionality in order to abstract the many supported cluster batch systems.
- Condor glide-in mechanism gathers Grid resources by using the pilot paradigm[4] and presents to the user a dynamic, virtual-private compute pool. As such, the monitoring limitations are the same as with a dedicated Condor pool. However, obtaining login access to the Grid resources will be significantly harder, if not impossible, as it requires direct negotiation with many administrators.

3. Pseudo-interactive monitoring using dedicated batch slots

Since true, unlimited interactive access to the running jobs does not seem either the easiest or the most acceptable option, our proposal is to use regular batch processing to solve the monitoring problem. The only requirements are that the monitoring job runs on the same worker node and with the same identity as the job that needs to be monitored, and that wait times are short, O(10) seconds at most. We call this **pseudo-interactive** monitoring.

3.1. Most monitoring needs are batch in nature

Before proposing a concrete solution, let's verify that most of the monitoring needs are indeed batch in nature. We thus identify six categories of frequently needed tasks performed by scientists while debugging a problem:

- **Looking at log files** – Most scientific applications log their progress in log files. Users are interested in looking at both the existence of such files as well as at their content. Most of the commands needed to achieve this can indeed be run in batch mode, e.g. *ls*, *cat* and *grep*. When true interactive access is needed, like with less, this can be simulated by first downloading the file via *cat*, or the batch system supported file transfer mechanism, and then running the interactive command locally.
- **Looking at data files** – This is a very similar use case as the one above, but may require specialized tools to digest the data. If the data file is small enough, downloading the file and then running the dedicated tool will work as above. If the file is too big, the tool can be run as a batch job, using the batch system file transfer mechanisms for delivery, if it does not require interactive capabilities. To the best of our knowledge, most scientific tools support at least limited functionality in batch mode.
- **Creating or modifying data files** – While debugging a problem, a user may want to change the content of a configuration or data file to unblock a stuck process. While true interactive editing, e.g. *vi*, is not possible, it may be simulated by using batch-friendly programs like *awk* and *sed*. If the file is not being changed and is small enough, it can also be downloaded, edited locally, and uploaded back, using the batch system file transfer mechanisms.
- **Looking at the status of the running processes** – Users are often also interested to see which processes are running and how much resources (e.g. CPU, RAM) they are using. The commands needed to achieve this, e.g. *ps* and *top*, can indeed be run in batch mode. More advanced commands, like getting the current stack trace or tracing the activity of a process over a limited period of time, are also possible with a limited amount of scripting. Only true interactive debugging, like stepping through the process execution using gdb, is not possible; however, the need for such activities is usually very rare as it can be approximated with other tools.
- **Sending signals to running processes** – If a process gets stuck, it may be preferable to kill just that process and let the job finish to get back partial results. Running a *kill* command is obviously possible in batch mode.
- **Looking at the status of the system** – Information about system (CPU and IO) load, memory usage, network traffic, etc. is often very valuable when trying to understand why jobs are taking longer than expected. The commands needed to achieve this can easily be run in batch mode.

As shown above, the vast majority of monitoring tools the users need can run as batch jobs. For the few that cannot, workarounds are often available. So using batch processing for monitoring is indeed a viable solution.

3.2. Implementing pseudo-interactive monitoring in Condor

Most Condor installations already run multiple batch slots per worker node, typically one **job batch slot** per CPU core (or thread). To implement pseudo-interactive monitoring, we propose to add one or more additional **monitoring batch slots** on each worker node, as shown in Fig. 2. Monitoring is light on resource usage; it doesn't consume many CPU cycles nor is it IO intensive. As such, adding additional batch slots to the worker nodes will not have any negative impact on the efficiency of real jobs. The Condor central manager, however, has to deal with many more batch slots, but in our experience this has not been a noticeable problem.

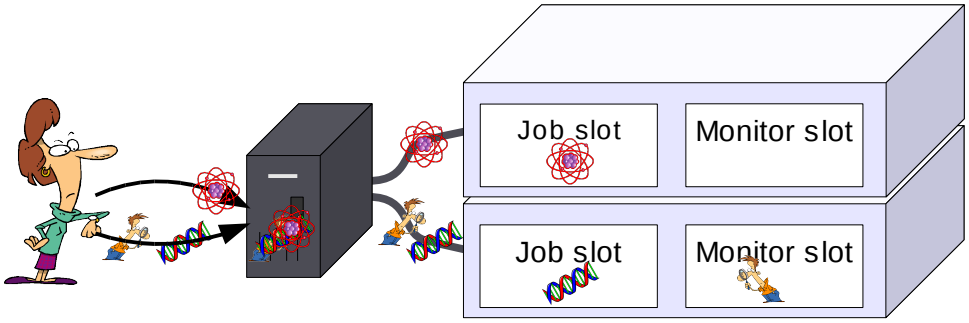

Figure 2: A monitoring batch slot provides pseudo-interactive monitoring

As we will monitor user jobs by submitting monitoring jobs, we need a way to distinguish between them. The easiest way is to define a dedicated attribute that monitoring jobs set (while the user jobs don't); the most obvious name for such an attribute is **`MonitoringJob`**.

In Condor, the number of batch slots is regulated by the **`NUM_CPUS`** setting[5]; by setting this to a number slightly higher than the number of physical CPU cores (or threads) we get the number of slots we need. On top of that, we need to split these batch slots into jobs slots and monitoring slots; we can do that by using **`NUM_SLOTS_TYPE_`***`X`* settings[5], paired with the appropriate **`START`** expression[5]. Consider the following simple example startd configuration:

```
REAL_NUM_CPUS = 8
MONITORING_SLOTS = 2
REAL_START_CONDITION = True
# Cannot use an expression for NUM_CPUS before Condor 7.3.1
# NUM_CPUS = $(REAL_NUM_CPUS) + $(MONITORING_SLOTS)
NUM_CPUS = 10
NUM_SLOTS_TYPE_1 = $(REAL_NUM_CPUS)
NUM_SLOTS_TYPE_2 = $(MONITORING_SLOTS)
START = ((SlotID<=$(REAL_NUM_CPUS)) && \
         ($(REAL_START_CONDITION))) \
      || ((SlotID>$(REAL_NUM_CPUS)) && \
         (MonitoringJob=?=True))
```

If the fraction of monitoring slots is significant, one may also want to explicitly minimize the amount of resources allocated to those slots, by tuning the **`SLOT_TYPE_`***`X`* setting[5] like in the example below:

```
# Monitoring slots should use only 1% of resources (but get the full slot)
SLOT_TYPE_2 = cpus=1, 1%
```

Additional protections in the startd configuration may also be desirable. For example, an administrator may want to limit the wallclock time of monitoring jobs to a few minutes. Or limit the memory usage to a few hundred MBs. Such limits can be implemented using the **`KILL`** setting[5]; a detailed description is however beyond the scope of this document.

On the submit side, regular user jobs don't need to be changed; by **not** defining the **`MonitoringJob`** attribute they will be limited to the proper job slots by the `START` expression.

Monitoring commands will need to be converted into batch jobs. The worker node on which the target user job is running also needs to be extracted; for example, by using `condor_q`. Such batch jobs will need to set the **`MonitoringJob`** attribute and specify in the requirements on which worker node they should run, as shown in the example fragment below:

```
# This tells the system this is a monitoring job
+JobMonitoring=True
# Assuming I discovered with condor_q that the job is running on node174.uni.edu
Requirements=(Machine=?="node174.uni.edu")
```

After submitting the monitoring job, it should start on the desired worker node following the next negotiation cycle; typically in less than one minute. The whole process is obviously too cumbersome to be performed by hand; to simplify basic pseudo-interactive monitoring, we are including a shell script called **`condor_monitor`**, analogous to `condor_run`, but specialized to the submission of monitoring jobs. The reader is welcome to expand it to accommodate the needs of his/her user community.

Unfortunately some problems remain. On the worker node the monitoring job will be running under the same UID as the target user job, assuming that the worker node is in the same UID_DOMAIN or GLEXEC is used[5]. However, if more than one job from that user is running on the worker node, it may be difficult to find the correct process ID. Similarly, the monitoring job will start under a different working directory than the target job; finding the correct working directory is non-trivial. We have no easy answers for the above problems in the generic case; additional assumptions, a try-and-error approach or some help from the user jobs are currently the best workarounds.

Finally, as mentioned already, the addition of monitoring slots increases the number of ClassAds that the central manager must consider when matching jobs to slots, so its memory usage and matchmaking time will increase. In addition to that performance cost, there is also an effect on the fair-share scheduler behavior. The addition of extra slots will cause the fair-share scheduler to calculate each user's fair share of the pool to be larger. Since the matchmaking algorithm sorts the users by priority from best to worst and then tries to give them each their fair share, the result is that users with better priority will tend to get a larger share of the normal job slots than they should. However, this is self-correcting over time, because users who use more machines will have their priority degrade and will therefore alternate between getting more than their fair share and then less than their fair share.

If either the performance cost or the scheduling quirk are a concern, one solution is to create a second central manager solely for the monitoring slots rather than having the normal and monitoring slots all managed by a single central manager. This is achievable in a round-about way by advertising all slots to both central managers and then using COLLECTOR_REQUIREMENTS to filter out the unwanted ClassAds[5] from the respective central managers.

3.3. Pseudo-interactive monitoring of Grid resources using Condor glide-ins

As mentioned in section 2, one way to access Grid resources is to create a virtual-private Condor pool by using glide-ins. As such, these resources look exactly like a regular Condor pool, so pseudo-interactive monitoring will work as described in the previous subsection.

One Grid WMS based on glide-ins is glideinWMS[6]. This system comes pre-configured with pseudo-interactive-friendly configuration, very similar to the one described above.

Moreover, since by default glidein-ins only use one job batch slot at a time, identifying the proper process tree and working directory is much simpler. The process tree of the glidein-in is very deterministic as shown in Fig. 3. By climbing the process tree up to the common Condor daemon, the user job process tree can be easily found. As shown in Fig. 4, Condor always starts the jobs as subdirectories of a predefined directory; by moving up to the parent directory, the user job working directory is the other child.

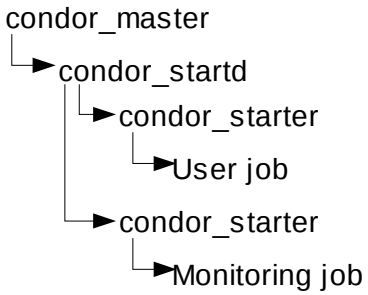

Figure 3: Glide-in process tree

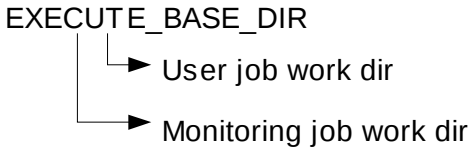

Figure 4: Glide-in work dir tree

It is also worth noting that glideinWMS provides user tools similar to the attached `condor_monitor` that automate the most commonly used monitoring tasks; i.e. ls, cat, ps, top and gdb stack dump. Users never need to know the details of the implementation.

3.4. Pseudo-interactive monitoring in other batch systems

We have not explored the feasibility of implementing a native monitoring-batch-slot solution for other batch systems. However, we are confident that it should be possible for most of them, since the underlying principles are very generic.

4. Conclusions

Distributed computing requires monitoring and debugging tools like any other kind of computing. However, most current workload management systems don't provide adequate tools for this task. We propose a simple solution based on dedicated monitoring batch slots and provide an actual implementation for the Condor batch system.

5. Acknowledgements

Fermilab is operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

This paper was partially supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357, and by the U.S. National Science Foundation grants PHY-0427113 (RACE) and PHY-0533280 (DISUN).

References

[1] B. Bode, D. M. Halstead, R. Kendall, and Z. Lei. “The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters”, In Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, October 2000.

[2] D. Thain, T. Tannenbaum, and M. Livny, "Distributed Computing in Practice: The Condor Experience" Concurrency and Computation: Practice and Experience, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[3] I. Foster and C. Kesselman, “The Grid: Blueprint for a New Computing Infrastructure”, San Francisco, CA: Morgan Kaufmann Publishers, 1998

[4] I. Sfiligoi, "Making science in the Grid world: using glideins to maximize scientific output," Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE , vol.2, no., pp.1107-1109, Oct. 26 2007-Nov. 3 2007

[5] Condor Team, “Condor Version 7.2 Manual”, http://www.cs.wisc.edu/condor/manual/v7.2/, Accessed May 2009

[6] I. Sfiligoi, “glideinWMS – A generic pilot-based Workload Management System”, Journal of Physics: Conference Series 119 (2008) 062044