Loadtest_condor – A workload generating framework for testing scalability and reliability of the Condor system

I. Sfiligoi and F. Würthwein
UC San Diego, La Jolla, CA 92093, USA
email: isfiligoi@ucsd.edu, fkw@ucsd.edu

Abstract: As any software product, Condor does not scale indefinitely, and needs to be scale tested in order to discover those limits. The loadtest_condor tool, developed by the Open Science Grid scalability team, provides a flexible framework to both generate the needed load on the system and analyze the results. This document provides a high level overview of the tool.

1. Introduction

Condor[1] is the main workload management system (WMS) used in Open Science Grid (OSG). It is being used as the batch system of many OSG Grid sites, as the Grid submission client[2] of OSG users, and as a OSG-wide overlay batch system[3]. Understanding its behavior in extreme conditions is thus very important.

OSG has a scalability team dedicated to testing of its critical software products, Condor being one of them. The OSG scalability team has thus been testing Condor in its various use cases for several years now.

Being a WMS, Condor scalability is linked to the number of jobs being handled by the system. To the best of our knowledge, there were no existing tools that would allow automated, and flexible submission of a large number of batch jobs, so the OSG scalability team developed one, and called it **loadtest_condor**. It is composed of two pieces; a load generating tool and a log analyzing tool.

2. Testing dimensions

While the number of jobs handled by the system is the single major metric, the type of jobs managed is just as important. We identified the following as being the most important characteristics; job state, job execution time, job coupling, and input and output sandbox size.

2.1. Job state

Each job in Condor has a state associated with it. When it is is submitted, it is assigned an *"Idle"* state. When it starts running, it enters the *"Running"* state. If something goes wrong, it is put into the *"Held"* state. Finally, when the job terminates, it enters the *"Completed"* state and then removed from the queue.

On top of the major state, each Condor job also has a sub-state. For example, a job with an input sandbox has a different sub-state before and after the sandbox is transferred to the execution side; a similar situation exists if a job produces an output sandbox. Another example is a Grid job; it needs to be delegated to the remote job queue, and the sub-state will be different before and after.

Knowing the state, and sub-state, of each job is important because Condor treats jobs in different (sub-)states very differently. In order to properly test a Condor system it is thus important to be able to specify the number of jobs in as many states as possible.

2.2. Job execution time

Switching between states is also an expensive task, so controlling how often this happen is imperative. And the average state switching rate is determined by the average number of running jobs divided by the mean job execution time, so any testing tool must be able to control the latter.

2.3. Job coupling

Knowing and controlling the average state switching rate is however not enough; understanding the distribution and the peak(s) is just as important.

The state switching distribution is tied both to job execution time and to job coupling. If all jobs are independent and with Gaussian execution times, the state switching distribution will be Gaussian as well. If the jobs are tightly coupled, e.g. they all terminate at the same time, the state switching distribution will be very spiky.

Controlling the job execution time distribution and job coupling is thus also very important.

2.4. Input and output sandbox size

One of the most expensive state transition operations is the transfer of input and output sandboxes, since it stresses both the IO system and the network. So any testing tool must be able to control it.

3. Load generating tool

The load generating tool, **loadtest_condor**, can control all of the above parameters. It is also able to submit jobs of different types, that will test the various Condor deployment use cases. The tool is command line based, and Figure 1 contains the supported options.

```
Usage:
  loadtest_condor.sh [options]
Where options is any combination of:
  -type <universe> [opts] (REQUIRED)
        grid gt2 <resource_name> : Submit Grid GT2 jobs
        grid gt5 <resource_name> : Submit Grid GRAM5 jobs
        grid cream <url> <batch> <queue> : Submit Grid CREAM jobs
        vanilla                  : Submit vanilla jobs
        local                    : Submit local jobs
  -req[uirements] : Requirements of the job
                    This maps to RSL for GT2 and REQUIREMENTS for vanilla jobs
  -end random|fixed|sync|int[erval] <val> (REQUIRED)
       random <seconds>   : Each job will sleep a random number of seconds
                            (val+-25%)
       fixed <seconds>    : Each job will sleep a fixed amount of seconds
       sync <unix_time>   : All jobs will try to end exactly at the same time
       interval <seconds> : All jobs will end at the next multiple of seconds
  -jobs      <count> : How many jobs to submit, default=10
  -clus[ter] <count> : How many jobs should I group per cluster, default=1000
```

```
  -maxidle <count>   : How many idle jobs are allowed (delay submission else),
                       default=-1 (disabled)
  -maxjobs <count>   : How many jobs are allowed in the queue
                       (delay submission else),
                       default=-1 (disabled)
  -in[file]  <file name> <file size> : Create an input file (default=no file)
  -out[file] <file name> <file size> : Create an output file (default=no file)
  -workdir <path>     : Where will the work directory be created,
                        default=current dir
```

*Figure 1: Command line options of **loadtest_condor***

Using the provided arguments, the tool creates test jobs and control files to be submitted to the Condor queue. Regulating the job pressure is delegated to Condor's dagman[4], avoiding the need for turning the tool into a daemon.

The tool itself is written in Python and has been tested on Redhat-compatible Linux based systems.

4. Analyzing the test results

Each time a job enters a new (sub-)state, Condor writes an event into a log file. By analyzing this log file, the tester can follow the behavior of the system.

Although the log file is easy to parse, having a tool that can summarize the information and present it in a well defined chronological order was found to be very useful. The OSG scalability team thus implemented one such tool and called it **log2timebins**.

The tool will aggregate based on a subset of the time string as formatted by Condor; for example, a 10 char substring will aggregate 10 minutes into a single bin. The values reported can be either the number of jobs in a specific (sub-)state, the number of state transitions or the combination of the two. An example run displaying state transitions can be found in Figure 2.

```
$ log2timebins.py -rates job.log 10
12/28 09:5 SB:  431 GB:    0 ST:  403 TM:    1 AB:    0
12/28 10:0 SB: 1081 GB:    0 ST:  704 TM:  407 AB:    0
12/28 10:1 SB: 1397 GB:    0 ST:  693 TM:  691 AB:    0
12/28 10:2 SB: 1409 GB:    0 ST:  718 TM:  703 AB:   20
12/28 10:3 SB: 1408 GB:    0 ST:  701 TM:  719 AB:    0
12/28 10:4 SB: 1289 GB:    0 ST:  582 TM:  705 AB:    0
12/28 10:5 SB: 1367 GB:    0 ST:  777 TM:  601 AB:    0
12/28 11:0 SB: 1193 GB:    0 ST:  433 TM:  759 AB:    0
12/28 11:1 SB:  426 GB:    0 ST:    0 TM:  425 AB:    0
```

*Fig. 2: Output of a **log2timebins** run.*

The acronyms are:

- *SB – Submitted*
- *GB – Grid submitted*
- *ST – Started*
- *TM – Terminated*
- *AB - Aborted*

4.1. External monitoring

The log analyzing tool provides only the information about the test jobs. The tester likely needs additional monitoring to understand the true state and health of the system.

There are several existing tools that already do this job, for example CondorView[5], procs_monitor[6] and CycleServer[7], so this package does not provide any additional system level monitoring.

5. Summary

In order to properly test the scalability limits of the Condor workload management system, the OSG scalability team has developed a workload generating framework composed of a load generating tool and a log analyzing tool. This framework has been used by the OSG scalability team for over a year to test various Condor versions, often pre-releases, allowing for the discovery of many scalability and reliability limits, many of which have since been fixed.

The framework, called **loadtest_condor**, was developed using an open source license and can be downloaded from sourceforge[8].

Acknowledgments

This work was partially sponsored by the US Department of Energy under Grant No. DE-FC02-06ER41436 subcontract No. 647F290 (OSG).

References

[1] “Condor Project Homepage”, http://www.cs.wisc.edu/condor/, Accessed December 2010.

[2] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", Journal of Cluster Computing volume 5, pages 237-246, 2002. http://dx.doi.org/10.1023/A:1015617019423

[3] I. Sfiligoi, D. C. Bradley,B. Holzman, P. Mhashilkar, S. Padhi, and F. Wurthwein, “The Pilot Way to Grid Resources Using glideinWMS”, Proceedings of Computer Science and Information Engineering, 2009 WRI World Congress on, pages 428 – 432, 2009. http://dx.doi.org/10.1109/CSIE.2009.950

[4] “DAGMan Applications”, http://www.cs.wisc.edu/condor/manual/v7.4/2_10DAGMan_Applications.html, Accessed December 2010.

[5] “CondorView”, http://www.cs.wisc.edu/condor/manual/v7.4/3_2Installation.html#sec:CondorView-Client-Install, Accessed December 2010.

[6] J. M. Dost and I. Sfiligoi, “Procs_monitor - A process-level resource monitor”, OSG-doc-1012, 2010. http://osg-docdb.opensciencegrid.org/cgi-bin/ShowDocument?docid=1012

[7] “CycleServer”, http://www.cyclecomputing.com/products/cycleserver/overview, Accessed December 2010.

[8] “OSG Scalability Area Tools”, http://sourceforge.net/projects/osgscal/, Accessed December 2010.