Minimizing draining waste through extending the lifetime of pilot jobs in Grid environments

I Sfiligoi[1], T Martin[1], B P Bockelman[2], D C Bradley[3], F. Würthwein[1]

[1]University of California San Diego, 9500 Gilman Dr, La Jolla, CA 92093, USA
[2]University of Nebraska – Lincoln, 118 Schorr Ctr, Lincoln, NE 68588, USA
[3]University of Wisconsin – Madison, 1150 University Ave, Madison, WI 53706, USA

isfiligoi@ucsd.edu

Abstract. The computing landscape is moving at an accelerated pace to many-core computing. Nowadays, it is not unusual to get 32 cores on a single physical node. As a consequence, there is increased pressure in the pilot systems domain to move from purely single-core scheduling and allow multi-core jobs as well. In order to allow for a gradual transition from single-core to multi-core user jobs, it is envisioned that pilot jobs will have to handle both kinds of user jobs at the same time, by requesting several cores at a time from Grid providers and then partitioning them between the user jobs at runtime. Unfortunately, the current Grid ecosystem only allows for relatively short lifetime of pilot jobs, requiring frequent draining, with the relative waste of compute resources due to varying lifetimes of the user jobs. Significantly extending the lifetime of pilot jobs is thus highly desirable, but must come without any adverse effects for the Grid resource providers. In this paper we present a mechanism, based on communication between the pilot jobs and the Grid provider, that allows for pilot jobs to run for extended periods of time when there are available resources, but also allows the Grid provider to reclaim the resources in a short amount of time when needed. We also present the experience of running a prototype system using the above mechanism on a few US-based Grid sites.

1. Introduction

In recent years, the pilot paradigm has become the dominant way of using widely distributed computing resources for the scientific communities, an example pilot product being the glideinWMS [1]. Its separation of resource provisioning from user job scheduling has proven to be very suitable for Distributed High Throughput Computing (DHTC) infrastructures, also known as Grid infrastructures, like the Open Science Grid (OSG) [2,3] and the European Grid Initiative (EGI) [4].

Traditionally, however, pilot jobs have only been managing one core, and thus a single user job at a time. With the proliferation of cores in modern processors this is not optimal anymore. There is increased pressure from the scientific communities [5,6] to be able to run multi-core user jobs, while at the same time allow single-core user jobs as well. The pilot jobs should thus be able to provision multiple cores at a time, and then carve them out to possibly several jobs in parallel.

The switch from single-job to multi-job scheduling has, however, significant implications on the pilot overhead. Pilot jobs by definition manage leased resources, and thus must return them to the resource providers by the time the lease expires. Since all user jobs are unlikely to terminate at the same time, this results in wasted computing due to the draining activity. This is especially wasteful when another pilot job from the same pilot owner is given access to the just-vacated resource again, as is often the case. The easiest way to reduce this overhead is to reduce the draining frequency by making the lease time as long as possible. A more detailed explanation is available in section 2.

Having very long leases is however problematic for most HTC resource providers, since it makes it difficult to effectively share resources among several user communities. Historically thus the leases have been kept relatively short. If leases were to be significantly extended, the resource providers will likely need to occasionally shorten those leases after jobs have already started, and this may result in some jobs being killed, as explained in section 3. The selection of target jobs has thus to be based on solid information to minimize waste, and pilot jobs should be given the option to gracefully terminate within the new limit, if at all possible, as explained in section 4.

Since there is currently no standard way for resource providers and pilot jobs to exchange meaningful information, in section 5 we propose a new, well defined mechanism that can achieve the desired goals. We also implemented a prototype system and ran it for over a month on a set of Grid sites, as explained in section 7.

2. Pilot jobs and draining cost

The basic premise of the pilot computing model is that it creates a dynamic, seemingly private batch system on top of resources leased from various sources. And, by definition, any lease eventually expires, so the pilot job managing the leased resource must go away before the deadline.

The lease time a pilot gets is typically enough to schedule and run several consecutive user jobs on that resources. However, eventually the remaining lease is too short to fit any further user job, and the pilot has no choice but prepare to go away. If the resource can only fit one job at a time, as is the case when only a single CPU core is being managed, the pilot can simply return the resource at that point; at virtually no cost. When, however, a pilot job is splitting the leased resources among several user jobs, it has to wait for all the already running jobs to finish first; at a substantial draining cost, as shown in figure 1.

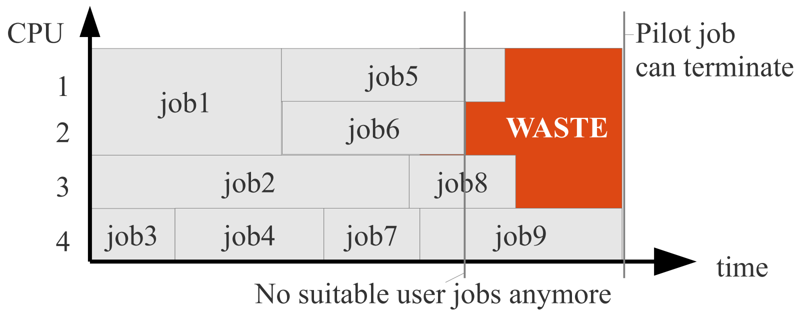

Figure 1. Limited pilot lease and job scheduling.

Minimizing this *absolute* draining cost is hard. The system has to work with the jobs users submit, and there may not always be jobs of perfect length in the queue at the right time. Moreover, knowing the actual runtime of any job is far from trivial; users are typically not very good estimators of job runtime [7] and automated prediction techniques have so far produced mixed results [8].

A way more manageable strategy is thus to minimize the *relative* draining cost of pilot jobs, by extending as much as possible the pilot job lifetimes, as opposed to scheduling several consecutive pilot jobs from the same pilot owner, each with a short lifetime as is the current practice. Since the average absolute draining cost is independent of the pilot job lifetime, the average relative draining cost is inversely proportionate with the average pilot job runtime.

3. The problem of long leases for resource providers

Historically, HTC resource providers have resisted providing long leases to users, including pilot providers. The reason for this attitude stems from the fact that it makes it difficult to effectively share resources among many users in the batch system environment.

To illustrate the problem, we here present a simplified use case. Let's assume that there are only two users in the HTC system, Alice and Bob, and that each owns half of the available job slots. At one point in time, Alice does not have any jobs to run, so Bob gets all the job slots for himself. However, an hour later, Alice decides that she wants to compute again, and submits a substantial number of jobs. Since there are no job slots available, Alice must wait for Bob's jobs to finish. If Bob's jobs were short, Alice likely would not mind the short wait. If, however, the wait stretches into days, Alice will most likely bitterly complain with the HTC system administrator.

So, for longer job leases, HTC systems have to implement preemption policies; i.e. forceful termination of jobs that are over-quota. And, traditionally, for most user jobs forceful termination meant losing all the work done up to that point, and thus resulting in significant waste of compute resources.

One further complication of long leases are misbehaving jobs, e.g. jobs that enter an infinite loop or end up waiting for extremely long time for data from remote network servers. In those cases, the job is effectively just wasting the CPU it has been given, and the sooner it is killed, the better. Traditionally, with short leases, this happens when the lease expires, keeping the waste reasonably low. If longer job leases were allowed, a reliable heartbeat mechanism would be needed to keep the waste within reasonable limits.

4. Smart termination of pilot jobs

The pilot jobs are however different from typical user jobs. The work performed in the pilot jobs is not all-or-nothing; every time a user job inside the pilot ends, the work performed by the resource is saved. Nevertheless, hard killing a pilot job is still wasteful; any work being performed by user jobs still running in the pilot will be lost, as shown in figure 2. Moreover, those pilot jobs may be running high priority jobs for the users, who will now have to re-start from the beginning somewhere else, and thus significantly delay the completion time.

Another notable difference is that the pilot jobs do not have to have a fixed runtime. Under most circumstances, pilot jobs can use as long a lease as they are given, but can also return the resource on short notice; i.e. as shown in figure 3, as soon as the last already running job finishes. This operation often also has a cost, as explained in section 2, but the user experience is significantly better.

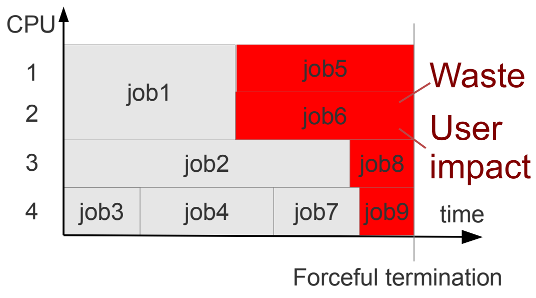

Figure 2. Waste resulting from killed pilots.

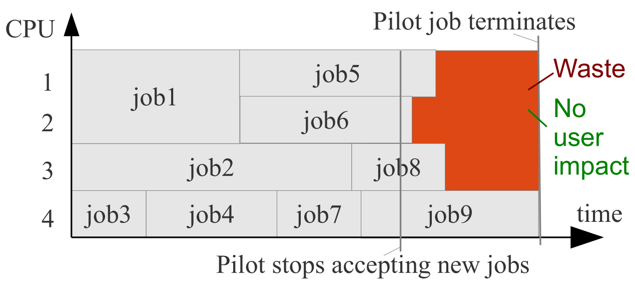

Figure 3. Pilot can terminate on short notice.

Resource providers should thus take these pilot job properties into consideration when implementing preemption policies. Unfortunately, at the time of writing, there was no standard way to either convey pilot job information to the resource owner, neither a way for the resource owner to request pilot draining, i.e. redefine the lease time. In this paper we thus propose a possible solution.

4.1. Information needed by the resource provider

When selecting the pilot job to preempt, the resource owner is likely to first select the over-quota pilot user with the least priority. This information is already available to the resource owner, so nothing new is needed in this step.

However, within the pilot jobs running by that pilot user, the resource provider will have to pick a subset of those pilot jobs; it should be very rare for the resource owner to preempt all pilot jobs for a pilot user. And to implement sensible policies, the resource provider will want to know:

a) The amount of time it will take for the pilot job to go away, if requested now
b) The amount of draining waste, if the pilot job was requested to go away now
c) The amount of waste that would result if the pilot job was hard killed now

A simple policy could use just one of the above values, but most resource providers are expected to use all three of them. An example policy could be: *If there are any pilots that can go away within 2 hours, pick the one with the least draining waste, else pick the one that is the least expensive to kill*.

One further policy optimization would consist in including information about the draining state of the considered pilot jobs. For example, if enough pilot jobs are already in draining state, no new pilot jobs need to be picked for termination. Of course, real-life policies will need protections in place to avoid abusing of this mechanism by the pilot owner to never get any pilot jobs killed.

4.2. Information available to the pilot job

Let's first analyze what information is actually available to the pilot job. Since it is managing the user jobs, it knows how many it has and how many CPU cores, out of the total leased ones it has allocated to them. The pilot knows when those user jobs have started, but should also have an estimate of when those user jobs are expected to end; it should have used that information to pick jobs that can fit in the remaining lease time.

Providing all the details to the resource provider is however not necessary. As described in the previous subsection, the resource provider really just needs three values. All of those values however change on a continuous basis, and are thus impractical to convey to the resource provider in a timely manner. Instead, we propose to convey aggregated information that only changes on discreet events, e.g. when one or more user jobs start or end, or the estimate of one or more user jobs' runtime is updated.

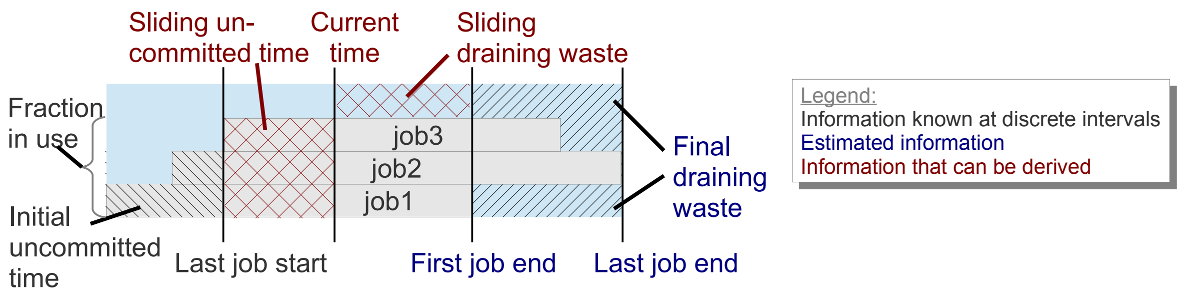

Figure 4. Information available to the pilot.

As seen in figure 4, the values that are known or estimated at discreet intervals are:

d) Fraction of resources in use
e) Last time a job started
f) The integral of resources that would be wasted, if the pilot would have been killed at that point
g) The estimated time the last job will end
h) The estimated time the first job will end
i) The estimated integral of resources that would be wasted, if draining started at that point

Knowing those values, and knowing the current time, two additional values can be derived:

j) The integral of resources that would be wasted between last job start and now, in case the pilot were to be killed
k) The integral of resources that would be wasted due to draining between now and the first job terminated, if draining were to be requested

The values needed by the resource provider can thus be calculated using the equations (3), (4) and (5).

$$j = (now - e) \cdot d \tag{1}$$

$$k = (h - now) \cdot (1 - d) \tag{2}$$

$$a = g - now \tag{3}$$

$$b = num\_cores \cdot (i + k) \tag{4}$$

$$c = num\_cores \cdot (f + j) \tag{5}$$

The pilot owner also may have some notion about the relative importance of this pilot job compared to all other pilot jobs he owns. As a simplified example, one pilot job may be serving only user jobs that have strict real-time requirements, while another would only be serving Monte Carlo simulation jobs. The effective cost of not getting the results in a timely manner from the former is obviously significantly higher, even if the amount of CPU time wasted is the same. The pilot thus would want the resource provider to take this into account in his preemption policy. We thus define one further value that the pilot can convey to the resource owner:

l) Relative value of this pilot job among all pilot jobs from this pilot user

Finally, the pilot knows if it has reached a point where it will stop starting new jobs, or not, and can report it to the resource owner. It could have entered that state either because the resource owner asked for it, or because the remaining lease time was not long enough to fit any of the available user jobs. We define this as:

m) Can the pilot job ever start new jobs that may finish after the currently running ones?

5. Defining a standard communication channel

The pilot job and the resource provider now need a standard communication channel. The proposal in this paper is to use a simple text file in the pilot job's startup directory, with each line being a "*attribute_name* = *attribute_value*" string.

The file used by the pilot job to communicate its information to the resource provider shall be named *.pilot.ad*. The syntax and semantics of the attributes is given in table 1.

Table 1. Attributes provided by the pilot jobs.

Attribute name	**Attribute type**	**Attribute semantics** (see definitions above)
LAST_JOB_START	Integer Representing UNIX time	*e*
FIRST_EXP_JOB_END		*h*
LAST_EXP_JOB_END		*g* (good faith estimate)
LAST_MAX_JOB_END		*g* (guaranteed max)
USED_FRACTION1k	Integer in the [0-1024] range Representing a fraction, with 1024 being 100%	*d*
ADD_UNCOM_TIME1k	Integer Representing a time integral multiplied by 1024	*f*
ADD_FINAL_EXP_WASTE1k		*i*
PRIORITY_FACTOR	Integer, higer is better	*l*
CAN_POSTPONE_LAST_JOB	Bolean, i.e. True or False	*m*

That same file will also serve as a heartbeat; the pilot job is expected to update the last modification time at least once an hour; while this is admittedly not completely fool proof, a stuck pilot is unlikely to perform this step, giving a reasonable assurance to the resource owner.

The file used by the resource provider to request pilot termination shall instead be named *.site.ad*. The syntax and semantics of these attributes are given in table 2. If the file does not exist, the pilot can assume the resource owner is not requesting draining.

Table 2. Attributes provided by the resource provider.

Attribute name	Attribute type	Attribute semantics
VACATE_DESIRED	Bolean, i.e. True or False	If True, the pilot should start draining
PAYLOAD_DEADLINE	Integer, representing UNIX time	The new lease expiration time (optional)

6. Analysis of the trust model

The information provided by the pilot to the resource provider is hard to verify. The resource provider should thus be cautious about trusting this information when making policy decisions.

Nevertheless, the recommended use of the pilot's provided information is to make a selection between jobs of that same user. So, if the pilot user decides to provide bogus information, it can only hurt itself, by having valuable pilot jobs terminated at the expense of expendable ones. Most intra-user selection policies based on this information are thus to be considered safe, but care should be taken if this information is used to calculate the grace period those pilots are given before being killed.

Using this information to pick between jobs from different pilot users is however strongly discouraged.

7. Implementing and operating a prototype

To validate our proposal, we configured the batch system at three USCMS Grid sites, UC San Diego, UN Lincoln and UW Madison, to read any produced .pilot.ad file for use by system administrators. All three sites use HTCondor [9] as their batch system, so the pilot provided information is propagated as ClassAds.

We have also created a prototype extension of glideinWMS [1], itself based on HTCondor, and deployed it on a subset of the production infrastructure in OSG. For about a month, a significant fraction of all glideinWMS pilots, aka glideins, directed to the above three Grid sites were of this prototype kind. The prototype glideins did advertise the needed attributes in the .pilot.ad file. They would also look for an eventual .site.ad, and enter draining state if requested.

The trial run was arguably quite successful. We were able to show that getting information from the pilot jobs is feasible, and that it is sufficient to make reasonable policy decisions. We also exercised in several occasions the resource provider to pilot job channel, i.e. creating the .site.ad file, to request draining of opportunistic pilot jobs that were over quota. This mechanism was very useful even for pilot jobs that started relatively recently, as it allowed us to obtain resources for high priority users much faster than we were able to do in the past.

In the process, however, we hit a few HTCondor bugs that created excessive load on the pilot system, so the trial run was temporarily stopped. The discovered bugs have been reported to the HTCondor developers, and should be fixed in HTCondor 8.0.4. We plan to start a second trial run, using the updated HTCondor binaries, shortly.

Unfortunately, the detailed description of the prototype setup would use up too much space in a paper, so we will not elaborate further. The interested reader can find the code used in the supplemental material section.

8. Related work

For the past year, the WLCG Workload Management Technical Evolution Group (WM-TEG) has been working on a proposal for passing information from the resource provider to the jobs [10]. Some of the authors of this paper have been occasionally involved in that work, and this paper has taken inspiration from it. However, the two activities are not in conflict, but are complementary.

The WM-TEG activity is focused mainly at providing static environment information to the jobs, so that they can discover the available resources in a uniform way, even when they are being scheduled in heterogeneous environments. There is some limited notion of dynamic information flow, namely the handling of scheduled maintenance periods, but that mechanism is not a good fit for conveying job specific commands to pilot jobs. More crucially, the WM-TEG activity does not specify any channel for a pilot job to provide information to the resource provider.

9. Conclusions

Unlike the single-core, single job use case, a pilot job handling several user jobs in parallel is bound to incur a significant draining cost at termination. Stopping pilot jobs just to have other pilot jobs from the same pilot owner re-scheduled on the same resource should thus be avoided as much as possible.

This implies having pilot jobs that on average have very long lifetimes. Unconditional long leases are however not acceptable for HTC resource providers, so there is a need for a mechanism for cleanly terminating a pilot job on short notice. Given the current lack of such a mechanism on deployed resources, in this paper we present one novel mechanism that has shown to be sufficient on a limited prototype deployment on top of production resources. The prototype run has also shown that this mechanism is beneficial for dealing with single core pilots as well.

Acknowledgements

This work was partially sponsored by the US National Science Foundation Grants PHY-1148698 and PHY-1120138.

References

[1] Sfiligoi I, Bradley D C, Holzman B, Mhashilkar P, Padhi S and Würthwein F 2009 *Comp. Sci. and Info. Eng., 2009 WRI World Congress on* **2** 428-432 doi:10.1109/CSIE.2009.950

[2] Pordes R et al. 2007 *J. Phys.: Conf. Ser.* **78** 012057 doi:10.1088/1742-6596/78/1/012057

[3] Sfiligoi I, Würthwein F, Dost J M, MacNeill I, Holzman B and Mhashilkar P 2011 Reducing the human cost of grid computing with glideinWMS *Proc. Cloud Computing 2011* Rome, Italy 217-221 ISBN 978-1-61208-153-3 http://www.thinkmind.org/index.php?view=article&articleid=cloud_computing_2011_8_40_20068

[4] Kranzlmüller D, Marco de Lucas J and Öster P 2010 *Remote Instr. and Virt. Lab.* 61-66 doi:10.1007/978-1-4419-5597-5_6

[5] Grandi C et al. 2012 *J. Phys.: Conf. Ser.* **396** 032053 doi:10.1088/1742-6596/396/3/032053

[6] Crooks D et al. *2012 J. Phys.: Conf. Ser.* **396** 032115 doi:10.1088/1742-6596/396/3/032115

[7] Bailey Lee C, Schwartzman Y, Hardy J and Snavely A 2005 *Job Scheduling Strategies for Parallel Processing L. N. in Comp. Sci.* **3277** pp 253-263 doi:10.1007/11407522_14

[8] Tsafrir D, Etsion Y and Feitelson D G 2007 *Parallel and Dist. Sys., IEEE Trans. on* **17 6** 789-803 doi:10.1109/TPDS.2007.70606

[9] Thain D, Tannenbaum T and Livny M 2005 *Concurrency and Computation: Practice and Experience* **17 2-4** 323-356 doi:10.1002/cpe.938

[10] The WLCG workload management technical evolution group 2013 Environment variables for multi-core jobs *CERN twiki* **Draft 0.98** https://twiki.cern.ch/twiki/bin/view/LCG/WMTEGEnvironmentVariables (Accessed Oct 2013)