The Instant Glidein; A generic approach for the late binding of jobs to various resource types

To cite this article: Laurence Field and Iain Steers 2017 J. Phys.: Conf. Ser. 898 092009

View the article online for updates and enhancements.

Related content

- PanDA: distributed production and distributed analysis system for ATLAS
  T Maeno

- The OSG Open Facility: an on-ramp for opportunistic scientific computing
  B Jayatilaka, T Levshina, C Sehgal et al.

- Photon emission and decay from generic Lorentz invariance violation
  H. Martínez-Huerta and A. Pérez-Lorenzana
The Instant Glidein: A generic approach for the late binding of jobs to various resource types

Laurence Field, Iain Steers
CERN, Geneva, Switzerland
E-mail: laurence.field@cern.ch

Abstract. High-throughput computing requires resources to be allocated so that jobs can be run. In a highly distributed environment that may be comprised of multiple levels of queueing, it may not be certain where, what and when jobs will run. It is therefore desirable to first acquire the resource before assigning it a job. This late binding approach has been implemented in resources managed by batch systems using the pilot jobs paradigm, with the HTCondor glidein being a reference implementation. For resources that are managed by other means such as the IaaS, alternative approaches for late binding may be required. This contribution describes one such approach, the instant glidein, as a generic method for implementing late binding to various resource types.

1. Introduction
Computational batch systems are a fundamental tool for high energy physics and other scientific domains. They enable computational tasks to be efficiently and seamlessly executed across computing resources and the area of task scheduling in particular has been an active research topic for computer science. One specific application is high-throughput computing (HTC), which is concerned with the number of tasks over a period of time that can be executed by computing resources. Attempts at HTC using multiple batch systems lead to the development of metaschedulers and ultimately to late binding approaches that aimed to overcome challenges relating to the multiple layers of scheduling. Recent technology advancements whereby the computing resources are delivered via an Infrastructure as a Service (IaaS) as virtual machines has brought another dimension and the advantages of late binding are also applicable.

This paper provides a brief overview of HTC and the emergence of metaschedulers in Section 2. The advantages of the late binding approach are discussed in Section 3 and the instant glidein is introduced as a reference implementation. Section 4 describes how the late binding approach can be applied to dynamic virtual resources and Section 5 presents the instant glidein as a simplified implementation.

2. High-Throughput Computing
Distributed processing systems enable computational tasks (jobs) to be executed on autonomous computing resources that are physically separated by a communication network, without the need of the task submitter to consider either the internal structure or load of the system[4]. The assignment of tasks to resources to optimize the utilization of those resources with respect to some selected metrics is delegated to the algorithms that the system implements[7]. This field
of task scheduling to optimize the utilization of distributed computing resources has been a rich area of computer science research for many years. With a general purpose system that supports multiple users, the demand on those resources is a stochastic property for both the arrival and execution times. This leads to the establishment of queues of waiting tasks that can lead to dynamic and continuous reconsideration of priorities. Demand in some cases can also be affected by resources that are dynamic and transient in nature. Early approaches to what was referred to as a load balancing algorithm [7] are today in referred to as computational batch systems and a general architecture is outlined as the LRMS (Local Resource Management System) in Figure 2.

![LRMS Diagram](image.png)

**Figure 1.** A general architecture of a LRMS

The term metacomputing has been used to describe such high-level systems that [15] enable many computing resources to be use as a single resource. The ambition is for the resulting metacomputing to be *seamless*, that using the resulting metacomputer should be as easy as using a single computing resource. One example of a metacomputing system was the Information Wide Area Year (I-WAY) experiment [15] that attempted to link 17 supercomputers into a single collaborative infrastructure to support computer science projects. With a metacomputing environment comprising of geographically-separated computing resources, each is typically under the control of a LRMS, a batch system. Metaschedulers have been proposed [17] to optimize the utilization of those computing resources based on state information provided by the LRMS. As the number of independent jobs submitted by user increases, the focus shifts from the performance of an individual job to the throughput over time of many jobs. Hence, the concern is with the number of operations per week or per month that can be extracted from the computing resources rather than the number of operations per second. This has been defined as HTC [6] and is the main driver behind many large-scale distributed computing initiatives. It originated as a class of computation problem from the high-performance computing (HPC) domain [15] and later was a significant application for metacomputing infrastructures composed of computing clusters based on commodity hardware. Grid computing extended the capability
of HTC by enabling tasks to executed on many different computing resources spanning multiple administrative boundaries. Again the main feature is those computing clusters can be seamlessly used from different locations. HTC suits applications that can be broken down into independent tasks which can be individually scheduled across the different computing resources. This drove the research into metaschedulers [14].

3. Late Binding
One of the issues with the metascheduling approach is that the decision of where the task is to be executed is made quite in advance. This means that the assignment is made without taking into consideration subsequent new information such as the computing resource halting the execution of new tasks due to either failure or higher-priority jobs. In addition the final execution environment is not known so even if the task starts, it may not complete successfully and hence adversely affects the turn around time as that task has to be rescheduled. As a result the metascheduling algorithm is not able to take into consideration information about the precise running conditions and last minute requests to the LRMS [10].

A solution to this is the concept of late binding in which a placeholder [11] task is used as resource reservation agent [10] which when executing on the computing resources, requests the most appropriate real task from a central task queue. This approach allows scheduling decision to be delayed until the the last possible moment and hence can take into consideration all information that is available at the time of the request. These agents can also perform basic sanity checks of the environment to avoid tasks being requested if the resource is not suitable. The result is that a private or application specific batch system is overlaid on the original infrastructure and is shown in Figure 4. One lesson learned from the experience of metascheduling is that resource allocation should be decoupled from task execution and the order respected [16].

An implementation of the late binding approach is the HTCondor Glidein [5]. The placeholder tasks are submitted using the standard HTCondor tooling. When the tasks executes on the computing resource it starts a HTCondor daemon process. The result is a private HTCondor pool that is created out dispersed computing resources by gliding-in HTCondor daemons. Using HTCondor Glideins requires a higher-level system to submit the Glideins and to handle their request for jobs when they execute on the computing resource. The glideinWMS [12] is an implementation of such as system that builds upon HTCondor itself to provide a central task queue for the HTCondor Glideins to contact.

4. Dynamic Computing Resources
The emergence of Virtual Machines delivered by Infrastructure as a Service (IaaS) has brought a new dimension to HTC [13]. Computing resources can now be dynamically provisioned on demand using resource leases which are typically defined in terms of wall time when using a pay-as-you-go model [3]. A potential benefit of this approach is that if short-term leases are used and the resources are recycled, the system can become self-healing by restoring the original state. Effects due to decay of the environment, such as the creation of temporary files, will be limited and any terminal issues can be cleaned before new resources are regenerated. As the potential capacity available from large-scale commercial providers can be significantly greater than the needs of the applications, this gives the appearance of infinite capacity. This would raise the question of whether queues and job scheduling are still required as all jobs could immediately be assigned a vacant resource. However, in reality the available capacity is finite due to other factors such as budget and price. The result is that the utilization of those available resources needs to be optimized with respect to some selected metrics and hence the algorithms that are implemented by batch systems are still required. As explained in the previous Section, the late binding approach is appropriate in this scenario to ensure that job execution occurs after the
resource has been allocated. The solution is therefore to overlay a batch system upon those dynamically provisioned computing resources. The remaining question is how to manage the lifecycle of those dynamic computing resources.

There are two main approaches. The VAC model [8] is where the lifecycle management occurs transparently for the user or application. From their perspective computing resources appear spontaneously and like particles in a physical vacuum they appear, potentially interact, and then disappear. The alternative is where the lifecycle management is done by the user or application themselves. This model will be defined as Throughput Amplification by the Stimulated Emission of Resources (TASER). Recent additions [9] to the glideinWMS serve as an example implementation of this model. It interacts with the IaaS to dynamically provision a computing resource to run the HTCondor Glidein. Figure 4 shows how the late binding approach can be used with either the VAC or TASER model with dynamic resources. Specifically the addition of feedback from the central job queue to the lifecycle manager changes the VAC model to the TASER model.

Figure 2. The VAC model with the TASER feedback indicated

5. The Instant Glidein

While provisioning the HTCondor Glidein for dynamic computing resources has proved to be successful, it can be further optimized. The original Glidein was designed to be pushed as job via a batch system to unknown heterogeneous resources. Hence a significant portion of the code handles the installation of HTCondor in user-space for many different platforms. It also has to handle differences in the configuration for HTCondor versions, how these interact with the platforms and what is required for the application.

Dynamically provisioning resources can reduce some of this diversity and results in a more homogeneous environment. By using a common VM image, the platform and also version of HTCondor can be known in advance. If a custom image is used HTCondor can also be installed as part of the operating system and hence removes the need to install it in user-space, further reducing what is required. The result is that only customizing the configuration and starting HTCondor to run as a short term lease is still required. We will define this shorter recipe as the Instant Glidein.
The CernVM [2] image has been selected as the base platform for this work. It is a small (17MB) image that delivers its contents just in time by means of the CernVM File System (CVMFS) [2]. CVMFS is a network file system with aggressive file level caching based on the HTTP/HTTPS protocols that is implemented locally using FUSE. It was developed so that a single virtual software appliance could be maintained as a common platform for multiple applications but each one could customize and feed such an appliance with their own software releases in an efficient way. The advantage of using CernVM for the Instant Glidein is that HTCondor is already available as part of the image.

To configure HTCondor as a short-term lease the following configuration is required.

**Listing 1.** HTCondor configuration for a short-term lease

```
Resource_Lease_Start_Time = 1474453218
Resource_Lease_Retire_Time = (Resource_Lease_Start_Time + 43200)
START = (time() < Resource_Lease_Retire_Time)
STARTD_NOCLAIM_SHUTDOWN = 300
MASTER_DAEMON_SHUTDOWN = (STARTD_StartTime =?= 0)
```

The Resource_Lease_Start_Time and Resource_Lease_Retire_Time parameters are used to define the times for when the lease starts and ends respectively. The START expression is used so that only new jobs will be fetched if the condition is true, i.e. that the lease is not ending. If no jobs are received after a period defined by the STARTD_NOCLAIM_SHUTDOWN period, then the HTCondor startd that fetches the jobs will terminated. When this occurs the MASTER_DAEMON_SHUTDOWN expression will ensure that HTCondor terminates.

HTCondor is started using the following code snippet.

**Listing 2.** Bash code snippet

```
pid_file="/var/run/condor/condor_master.pid"
/usr/sbin/condor_master -f -pidfile ${pid_file} &
sleep 5 # Need to wait for pid file to be created
if [ ! -e ${pid_file} ]; then
    wait $(cat ${pid_file})
fi
/sbin/shutdown
```

The main HTCondor process (condor_master) is forked and the process that executed the fork waits for HTCondor process to terminate. On termination the machine is shutdown. The lifecycle manager can then watch for virtual machines in the shutdown state and clean them up before new virtual machines are created. The Instant Glidein is being used to provide a common platform for LHC@home where BOINC [1] is used as lifecycle manager.

### 6. Conclusion

This paper provided a brief overview and the origins of HTC. It highlighted the use of metaschedulers to consolidate multiple batch systems into one seamless computational resource. The limitation of metascheduling was explained and hence why late binding is required to ensure that job allocation occurs after resource allocation. Two models for delivering dynamic computing resources via an Infrastructure as a Service (IaaS) as virtual machines were introduced and it was shown how the these models use late binding. Specifically the TASER model was defined to complement the VAC model and describes the scenario where throughput is controlled by directly controlling the creation of computing resources. Finally, the instant glidein was introduced as a simplification of the HTCondor glidin mechanism to overlay a batch system upon dynamic computing resource provided as virtual machines.
References


