Journal of Physics: Conference Series **331** (2011) 062039 doi:10.1088/1742-6596/331/6/062039

Job submission and control on a generic batch system: the BLAH experience.

Massimo Mezzadri, Francesco Prelz and David Rebatto

INFN, Sezione di Milano, via G. Celoria 16, I-20133 Milano

E-mail: blah@mi.infn.it

Abstract. The status, functionality and design rationale of the 'Batch Local Ascii Helper' (BLAH) service for submitting and controlling jobs on a batch system are presented. BLAH is part of both the Condor and CREAM systems. At the end of the major cycle represented by the EU EGEE projects, reflexion on its evolution provides some insight on technological choices that can stand the proof of time.

1. Introduction

By the end of the EU Datagrid project in 2003, it had become clear that the frameworks for remote job submission and control that had been adopted for the prototype distributed computing infrastructures (such as the Globus GRAM [1] implementations), were being limited mostly by scaling and performance issues. Several workarounds to allow the control of O(10000) jobs on existing production batch systems were implemented at that time, some of which are still deployed today in the running EGI [13] and WLCG [14] infrastructures. At the same time, the need emerged

- to isolate and focus on individual components of the job submission and control framework;
- to rationalise and integrate what was learned in the production environment.

One of the subsystems that was identified, where many ad-hoc overload prevention measures had already been introduced, was the *local* submission and control of a job received from a remote user to be scheduled and executed on a batch system. This process included the steps of local identity mapping based on grid-wide credentials, and the forwarding and renewal of such valid (X509) user credentials.

Development of the Batch Local ASCII helper (BLAH) started in 2004, with two reference applications: the newly developed CREAM job submission and control web service [2], where it would serve as the main job transfer path, and the well established Condor system [3], where it would serve as the interface to delegate the execution of Condor jobs to another third-party batch system.

As the main purpose of the component we describe is to address many *practical* issues that arise when communication with third-party batch system implementations is established, many original design intentions (such as developing a truly *stateless* system) had to be dropped as the system commissioning and scaling-up progressed. We see the experience gained in the field as invaluable, so we aim to describe here both the functionality of the BLAH component and at least some of the lessons learned in the development process.

(optional) **Identity mapping service** via either `glexec` [7] or `sudo`
Batch system abstraction layer - Submit jobs - Get status of submitted jobs - Control (hold, release, cancel) jobs
(optional) **X509 Proxy Renewal**

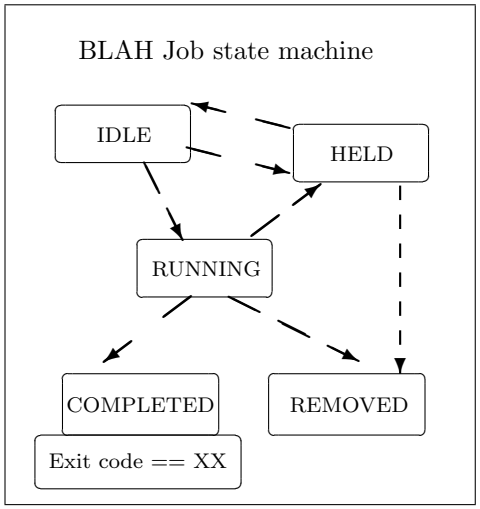

Table 1. These job submit and control operations, and the corresponding job state machine transitions, are matched and applied to all supported batch systems in BLAH.

2. The BLAH abstraction layer

As outlined in the introduction, BLAH provides a minimal common functional abstraction of batch systems, supporting three fundamental operations and a simplified job state diagram. These are described in Table 1.

2.1. Supported batch systems

At the time of writing, the BLAH package includes scripts and tools that communicate with the following batch systems:

1. Platform LSF ([8])
2. Torque/PBS ([9])
3. Condor ([3])
4. SUN/Oracle Grid Engine ([10])
   (this development courtesy of EGEE collaborators at LIP in Portugal and PIC in Spain).

Other systems can be supported: the involvement of groups with significant *production* experience on new candidates for support is essential.

2.2. A time-honored text command protocol

Given the specialized scope of the BLAH package, *ease of hands-on troubleshooting* was considered of primary importance. This led to prefer a classical text command protocol over other forms of interaction with the service (SOAP, Web services and other forms of APIs). Condor, one of our reference applications, had already such an interface (the GAHP, [5]) in place to communicate with standalone interfaces to third-party Grid infrastructures, so we adopted a compatible protocol. Text commands can be exchanged via the standard streams or a network socket. Given the local scope of the BLAH application, the latter is of very limited practical use.

A list of the main commands that form the BLAH protocol (BLAHP) can be found in Table 2, while the main attributes for job description at submit time are listed in Table 3.

`COMMANDS` List all the commands which are implemented by the current instance of the BLAH server.
`BLAH_SET_GLEXEC_DN` <*user proxy*> <*ssl client certificate*> <*prevent limiting proxy ?*> Enables local user mapping via `glexec` for all subsequent actions on jobs. The call is synchronous and the effect is immediate.
`BLAH_SET_SUDO_ID` <*requested user id*> Enables local user mapping via `sudo` for all subsequent actions on jobs. The call is synchronous and its effects immediate.
`BLAH_JOB_SUBMIT` <*request id*> <*submit classad*> Submit a job request to the local batch system, in the queue specified in the submit classad. See table 3 for a summary of attributes allowed in the submit classad.
`RESULTS` Get all available results for pending asynchronous commands.
`BLAH_JOB_STATUS` <*request id*> <*job ID obtained from submit*> Query and report the current status of a submitted job.
`BLAH_JOB_REFRESH_PROXY` <*request id*> <*job ID*> <*path of to fresh proxy file*> Renew the proxy of an already submitted job. The job has to be in IDLE, RUNNING or HELD status. The proxy is forwarded to the worker node for RUNNING jobs.
`BLAH_JOB_CANCEL` <*request id*> <*job ID obtained from submit*> Attempts removal of an IDLE job request, or termination of all processes associated with a RUNNING job, releasing any associated resources.
`BLAH_JOB_HOLD` <*request id*> <*job ID obtained from submit*> Put an IDLE job request in a HELD status, preventing its execution. If the job is already running RUNNING it can be HELD too, depending whether the underlying batch system supports this feature.
`BLAH_JOB_RESUME` <*request id*> <*job ID obtained from submit*> Resume execution of a previously HELD job.
`VERSION` Return the version string for the current instance of BLAH and the current version of the protocol.
`QUIT` Request immediate termination of the current BLAH instance.

Table 2. Essential BLAH commands.

3. Why did BLAH fail at remaining a *stateless* service ?

One possible design option for a layer like the one we are describing would be to have it purely and transparently pass messages to and from the underlying batch system, making it the perfect candidate for a *stateless* system, or perhaps a pure interface or API specification.

We note right away that an effort to provide a common batch system API has already been carried out within the 'Open Grid Forum' by the "Distributed Resource Management Application API" (DRMAA) group ([6]).

The point we'd like to cover here is that reference applications of the DRMAA interface (or of any pure common API for for job management) usually implement the API on top of existing job management services, so the API inherits the performance characteristics of the underlying batch system. We noted in the Introduction that our effort aims to recollect and integrate *practical* experiences made with existing instances of production batch systems. These often include accessing the batch system via various levels of indirection to avoid triggering system

Cmd	Full path of the executable in the local filesystem.
Args	List of individual arguments for the executable (no `/bin/sh` convention on argument separation, but separate arguments).
In	Full path in the local filesystem where the standard input for the executable is found.
Out	Full path in the local filesystem where the standard output of the executable will be stored (at job completion).
Err	Full path in the local filesystem where the standard error of the executable will be stored (at job completion).
X509UserProxy	Full path where the proxy certificate is stored.
Env	Semicolon-separated list of environment variables of the form: `<parameter> = <value>`.
Stagecmd	Determines if the executable of the job must be copied on the worker node: can be 'TRUE' or 'FALSE'.
Queue	Queue in the local batch system where the job must be enqueued.
Gridtype	String indicating the underlying local batch system, as more than one system can be supported by the same instance of BLAH.
CERequirements	Classad logical expression that can be used to pass attributes to a shell script callout used to change submit options on a per-job basis.

Table 3. Main attributes for job description. These are expressed in the syntax of Condor 'classified ads' (classads) [4].

overloads (e.g. by concurrent requests for multiple job status). We also faced the practical issue of integrating with the X509 authentication infrastructure that was used in production by our main funding project (EGEE [12]). This typically requires special handling of the X509 user proxy certificates, and the need of renewing them while a job is running. This capability often exceeds the available file transfer options of the batch systems we support.

Adapting to these needs *and* scaling the service up (in terms of numbers of concurrent jobs handled) to exceed current production levels forced us to introduce job state and several ancillary services to our system. The structure of these extra services is depicted in Figure 1, that we'll now briefly comment.

The most frequent operation that an intermediate layer such as BLAH has to perform, both directly on behalf of its clients and to check on the feasibility of certain requests, is to obtain the status of a submitted job. Experience has shown that most batch system cannot handle the rate of automated requests for the status of individual jobs when these are serviced synchronously out of the batch system central job data repository. Two indirect approaches, that were both implemented in BLAH, can help here:

1) Job state changes can be captured and obtained by checking on appends to the batch system log or journal files. These files are often accessible only on the batch system central server, by a properly privileged administrative user. The *BLAH Log Parser* process takes care of maintaining a cached view of job status based on information found in the log files, that can be queried for as needed by BLAH processes.
2) A complementary approach, that doesn't normally require the privileges of users other than the one running BLAH, throttles requests for the status of *all* jobs at a fixed rate, and maintains a persistent cache of the job status. This is handled in BLAH by the *BLAH Updater*, feeding a BLAH Job Registry on local disk.

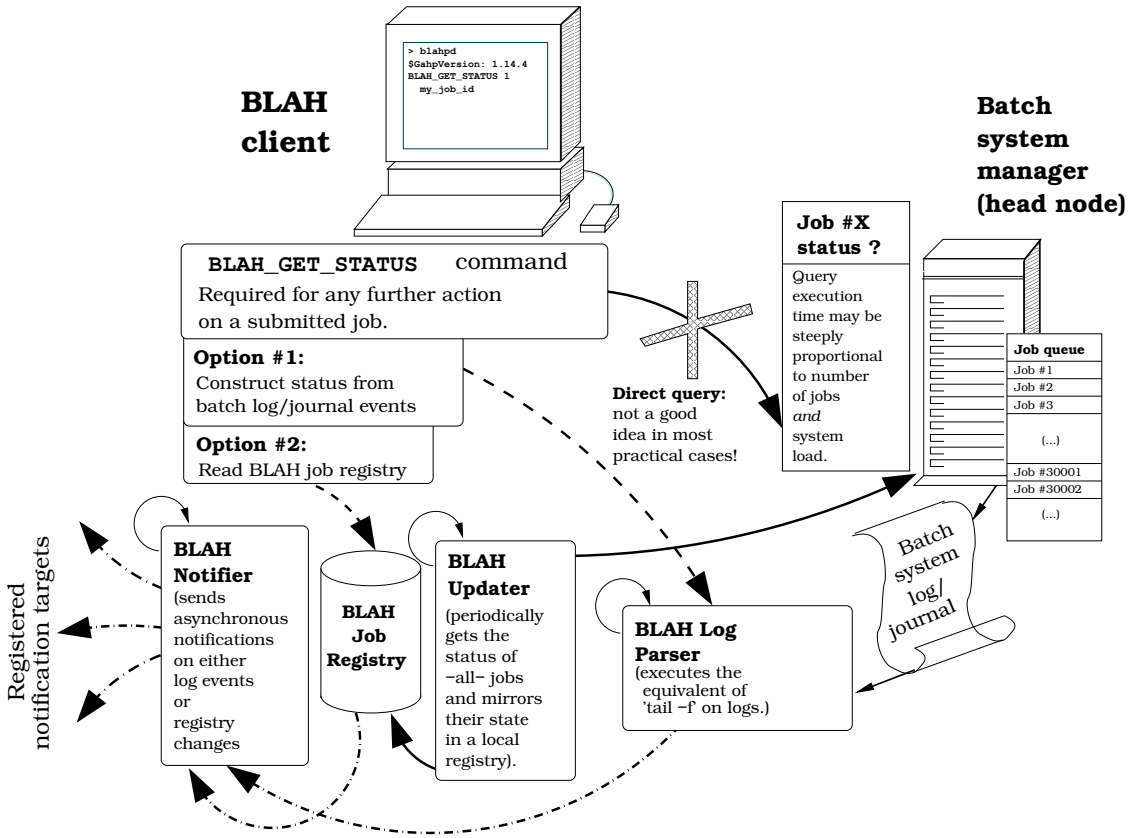

Figure 1. Current state of the BLAH services. Scale, efficiency and deployment consideration required the introduction of several levels of indirection.

The BLAH Job Registry stores job data in direct access format. Processes that read it usually maintain an index cache ordered by job ID for fast searches. The recent incident of a production instance with 40 concurrent BLAH processes causing memory thrashing when caching the indices of 120000 monitored jobs prompted the addition of `mmap()` based sharing of the job index.

The pressure of job status requests can also be kept under control by asynchronously pushing job status changes back to interested clients. This is achieved by the *BLAH Notifier* process, that can be fed either by events from log parsing, or by job status changes in the registry. This approach is used in particular in the BLAH-CREAM [2] integration.

On systems that lack dedicated support for X509 proxy renewal for running jobs, this task is achieved by *BLAH Proxy Renewal* server processes that are started in parallel with jobs on execution nodes.

4. Conclusions

With the end of the series of EU EGEE projects, a major cycle in the development and deployment of software for distributed job execution came to an end. As it was attempted to do at the end of the EU Datagrid project [11], we try to express here a few of the practical lessons learned, in the hope they will not be forgotten while the next technological cycle is in progress:

- **Most issues when interfacing to third-party systems are implementation issues**, and cannot be set aside by careful interface design. When trying to address external job

information access bottlenecks in the context of the design of individual batch systems, usually it is found that this conflicts with the requirement of speedy read-locked access to the same information needed for the smooth operation of the batch system scheduler. The addition of a few layers of indirection in the *implementation* appears to be the the only viable way forward here.

- **Semantical richness doesn't automatically translate into syntactical complexity**. We found a traditional, easy-to-troubleshoot text communication protocol a good and sustainable match to our needs.
- **Requests should not be allowed to queue and accumulate where they have an impact on system load**. This has been known for a long time, yet it is still too easy to bring job management systems to a halt by submitting *many* legitimate and properly authorised requests. Proper timeouts, throttles and resource quotas have to be implemented for *every* operation.
- **Even if job status access is made faster, it will eventually saturate**. Moving the threshold of sustainable job status access up from O(10000) concurrent jobs almost immediately required an effort to scale to the next order of magnitude, O(100000) jobs: every optimization path that is opened barely allows to move to the next challenge.

References

[1] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. *"A Resource Management Architecture for Metacomputing Systems"*. In The 4th Workshop on Job Scheduling Strategies for Parallel Processing, pages 62-82. Springer-Verlag LNCS 1459, 1998.

[2] C. Aiftimiei, P. Andreetto, S. Bertocco, S. Dalla Fina, A. Dorigo, E. Frizziero, A. Gianelle, M. Marzolla, M. Mazzucato, M. Sgaravatto, S. Traldi, L. Zangrando, *"Design and Implementation of the gLite CREAM Job Management Service"*, Future Generation Computer Systems, Volume 26, Issue 4, April 2010, pp. 654-667

[3] Michael Litzkow, Miron Livny, and Matt Mutka, *"Condor - A Hunter of Idle Workstations"*, Proceedings of the 8th International Conference of Distributed Computing Systems, pages 104-111, June, 1988. Website: `http://www.cs.wisc.edu/condor`.

[4] Rajesh Raman, Miron Livny, and Marvin Solomon, *"Matchmaking: Distributed Resource Management for High Throughput Computing"*, Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL.
See also `http://www.cs.wisc.edu/condor/classad`.

[5] Reference of the Grid ASCII Helper protocol: `http://www.cs.wisc.edu/condor/gahp/`

[6] Open Grid Forum DRMAA specifications GFD.22 and GFD.130, see `http://www.drmaa.org`.

[7] D. Groep, O. Koeroo and G. Venekamp, *"gLExec: gluing grid computing to the Unix world"*. presented to the CHEP 2007 conference, J. Phys. Conf. Ser. 119(2008)062032

[8] Platform LSF information is currently available at this site:
`http://www.platform.com/workload-management/high-performance-computing/lp`

[9] Torque resource manager information is currently available at this site:
`http://www.clusterresources.com/products/torque-resource-manager.php`

[10] Sun/Oracle Grid Engine information is currently available at these sites:
`http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html`
`http://gridengine.sunsource.net/`.

[11] DataGrid WP1 members (G. Avellino *et al.*), *"The first deployment of workload management services on the EU DataGrid Testbed: feedback on design and implementation."* Presented at the CHEP 2003 Conference, La Jolla.

[12] Enabling Grids for E-science in Europe `http://www.eu-egee.org/`

[13] The European Grid Initiative, `http://www.egi.eu/`

[14] The Worldwide LHC Computing Grid, `http://lcg.web.cern.ch/LCG`