EUROPEAN MIDDLEWARE INITIATIVE

LOGGING AND BOOKKEEPING – USER'S GUIDE

Document version:	**1.2.9**
EMI Component Version:	**3.x**
Date:	**June 21, 2011**

This work is co-funded by the European Commission as part of the EMI project under Grant Agreement INFSO-RI-261611.

CONTENTS

L&B DOCUMENTATION AND VERSIONS OVERVIEW

The Logging and Bookkeeping service (L&B for short) was initially developed in the EU DataGrid project[1] as a part of the Workload Management System (WMS). The development continued in the EGEE, EGEE-II and EGEE-III projects,[2] where L&B became an independent part of the gLite[3] middleware [1], and then in the EMI Project.[4]

The complete L&B Documentation consists of the following parts:

- **L&B User's Guide** - this document. This user's guide explains how to use the Logging and Bookkeeping (L&B) service from the user's point of view. The service architecture is described thoroughly. Examples on using L&B event logging command to log a user tag and change job ACL are given, as well as L&B query and notification use cases.
- **L&B Administrator's Guide** [2]. This administrator's guide explains how to administer the Logging and Bookkeeping (L&B) service. Several deployment scenarios are described together with the installation, configuration, running and troubleshooting steps.
- **L&B Developer's Guide** [3]. This developer's guide explains how to use the Logging and Bookkeeping (L&B) service API. Logging (producer), querying (consumer) and notification API as well as the Web Services Interface is described in details together with programing examples.
- **L&B Test Plan** [4]. This test plan document explains how to test the Logging and Bookkeeping (L&B) service. Two major categories of tests are described: integration tests (include installation, configuration and basic service ping tests) and system tests (basic functionality tests, performance and stress tests, interoperability tests and security tests).

The following versions of L&B service are covered by these documents:

- **L&B version 3.0**: included in the EMI-1 Kebnekaise release
- **L&B version 2.1**: replacement for **L&B version 2.0** in gLite 3.2,
- **L&B version 2.0**: current stable version, in production as part of gLite 3.2,
- **L&B version 1.x**: old stable version, in production as part of gLite 3.1.

The older version of L&B that appeared in gLite 3.0 became obsolete and is not maintained anymore.

L&B packages can be obtained from two distinguished sources:

- **gLite releases**: gLite node-type repositories, offering a specific repository for each node type such as *glite-LB*
- **emi releases**: EMI repository or EGI's UMD repository, offering all EMI middleware packages from a single repository and relying on EPEL for dependencies

[1] http://eu-datagrid.web.cern.ch/eu-datagrid/
[2] http://www.eu-egee.org/
[3] http://www.glite.org
[4] http://www.eu-emi.eu/

Note: Despite offering the same functionality, binary packages obtained from different repositories differ and switching from one to the other for upgrades may not be altogether straightforward.

Updated information about L&B service (including the L&B service roadmap) is available at the L&B homepage: http://egee.cesnet.cz/en/JRA1/LB

1 L&B ARCHITECTURE

L&B's primary purpose is tracking WMS jobs as they are processed by individual Grid components, not counting on the WMS to provide this data. The information gathered from individual sources is collected, stored in a database and made available at a single contact point. The user gets a complete view on her job without the need to inspect several service logs (which she may not be authorized to see in the entirety or she may not be even aware of their existence).

While L&B keeps track of submitted and running jobs, the information is kept by the L&B service also after the job has been finished (successfully completed its execution, failed, or has been canceled for any reason). The information is usually available several days after the last event related to the job arrived, to give user an opportunity to check the job's final state and eventually evaluate failure reasons.

As L&B collects also information provided by the WMS, the WMS services are no longer required to provide a job-state querying interface. Most of the WMS services can be even designed as stateless—they process a job and pass it over to another service, not keeping state information about the job anymore. During development and deployment of the first WMS version this approach turned to be essential in order to scale the services to the required extent [5].

L&B must collect information about all important events in the Grid job life. These include transitions between components or services, results of matching and brokerage, waiting in queue systems or start and end of actual execution. We decided to achieve this goal through provision of an API (and the associated library) and instrumenting individual WMS services and other Grid components with direct calls to this API. But as L&B is a centralized service (there exists a single point where all information about a particular job must eventually arrive), direct synchronous transfer of data could have prohibiting impact on the WMS operation. The temporary unavailability or overload of the remote L&B service must not prevent (nor block) the instrumented service to perform as usual. An asynchronous model with a clear *asynchronous delivery semantics*, see Sect. 1.1.2, is used to address this issue.

As individual Grid components have only local and transient view of a job, they are able to send only information about individual events. This raw, fairly complex information is not a suitable form to be presented to the user for frequent queries. It must be processed at the central service and users must be presented primarily with this processed form. This form is derived from the *job state* and its transition, not from the job events themselves. The raw information is still available, in case more detailed insight is necessary.

While the removal of state information from (some of) the WMS services helped to achieve the high scalability of the whole WMS, the state information is still essential for the decisions made within the resource broker or during the matchmaking process. For example decision on job resubmission is usually affected by the number of previous resubmission attempts. This kind of information is currently available in the L&B only, so the next "natural" requirement has been to provide an interface for WMS (and other) services to the L&B to query for the state information. However, this requirement contains two contradictions: (i) due to the asynchronous event delivery model, the L&B information may not be up to date and remote queries may lead to unexpected results (or even inconsistent ones—some older information may not be available for one query but may arrive before a subsequent query is issued), and (ii) the dependence on a remote service to provide vital state information may block the local service if the remote one is not responding. These problems are addressed by providing *local view* of the L&B data, see Sect. 1.1.6

1.1 CONCEPTS

1.1.1 JOBS AND EVENTS

To keep track of user jobs on the Grid, we first need some reliable way to identify them. This is accomplished by assigning a unique identifier, which we call *jobid* ("Grid jobid"), to every job before it enters the Grid. A unique jobid is assigned, making it the primary index to unambiguously identify any Grid job. This jobid is then passed between Grid components together with the job description as the job flows through the Grid; the components themselves may have (and usually do) their own job identifiers, which are unique only within these components.

Every Grid component dealing with the job during its lifetime may be a source of information about the job. The L&B gathers information from all the relevant components. This information is obtained in the form of L&B events, pieces of data generated by Grid components, which mark important points in the job lifetime (for example passing of job control between the Grid components are important milestones in job lifetime independently on the actual Grid architecture); see Appendix A for a complete list. We collect those events, store them into a database and simultaneously process them to provide higher level view on the job's state. The L&B collects redundant information—the event scheme has been designed to be as redundant as possible—and this redundancy is used to improve resiliency in a presence of component or network failures, which are omnipresent on any Grid.

The L&B events themselves are structured into *attribute* = *value* pairs, the set of required and optional attributes is defined by the event *type* (or scheme). For the purpose of tracking job status on the Grid and with the knowledge of WMS Grid middleware structure we defined an L&B schema with specific L&B event types (see AppendixA). The schema contains a common base, the attributes that must be assigned to every single event. The primary key is the jobid, which is also one of the required attributes. Among other common attributes belong currently the timestamps of the event origin and of the event arrival to LB, generating component name, the event type, its priority and sequence code (see Sect. 1.1.3) and so on. For a complete list of attributes see [3].

While the necessary and sufficient condition for a global jobid is to be Grid-wide unique, additional desired property relates to the transport of events through the network: All events belonging to the same job must be sent to the same L&B database. This must be done on a per message basis, as each message may be generated by a different component. The same problem is encountered by users when they look for information about their job—they need to know where to find the appropriate L&B database too. While it is possible to devise a global service where each job registers its jobid together with the address of the appropriate database, such a service could easily become a bottleneck. We opted for another solution, to keep the address of the L&B database within the jobid (actually, fully qualified hostname is strongly recommended instead of numeric address and numeric IPv6 address is not supported for backward compatibility reasons). This way, finding appropriate L&B database address becomes a local operation (at most parsing the jobid) and users can use the same mechanism when connecting to the L&B database to retrieve information about a particular job (users know its jobid). To simplify the situation even further, the jobid has the form of an URL, where the protocol part is "https", server and port identify the machine running the appropriate L&B server (database) and the path contains base64 encoded MD5 hash of random number, timestamp, PID of the generating process and IP address of the machine, where the jobid was generated. Jobid in this form can be used even in the web browser to obtain information about the job, provided the L&B database runs a web server interface. This jobid is reasonably unique—while in theory two different job identifications can have the same MD5 hash, the probability is low enough for this jobid to represent a globally unique job identification.

1.1.2 EVENT GATHERING

As described in the previous section, information about jobs are gathered from all the Grid components processing the job in the form of L&B events. The gathering is based on the *push* model where the components are actively producing and sending events. The push model offers higher performance and scalability than the pull model, where the components are to be queried by the server. In the push model, the L&B server does not even have to know the event sources, it is sufficient to listen for and accept events on defined interface.

The event delivery to the destination L&B server is asynchronous and based on the store–and–forward model to minimize the performance impact on component processing. Only the local processing is synchronous, the L&B event is sent synchronously only to the nearest L&B component responsible for event delivery. This component is at the worst located in the same local area network (LAN) and usually it runs on the same host as the producing component. The event is stored there (using persistent storage – disk file) and confirmation is sent back to the producing component. From the component's point of view, the send event operation is fast and reliable, but its success only means the event was accepted for later delivery. The L&B delivery components then handle the event asynchronously and ensure its delivery to the L&B server even in the presence of network failures and host reloads.

It is important to note that this transport system does not guarantee ordered delivery of events to the L&B server; it *does* guarantee reliable and secure delivery, however. The guarantees are statistical only, as the protocol is not resilient to permanent disk or node crashes nor to the complete purge of the data from local disk. Being part of the trusted infrastructure, even the local L&B components should run on a trusted and maintained machine, where additional reliability may be obtained for example by a RAID disk subsystem.

1.1.3 EVENT PROCESSING

As described in the previous section, L&B gathers raw events from various Grid middleware components and aggregates them on a single server on a per-job basis. The events contain a very low level detailed information about the job processing at individual Grid components. This level of detail is valuable for tracking various problems with the job and/or the components, and as complementary events are gathered (for example each job control transfer is logged independently by two components), the information is highly redundant. Moreover, the events could arrive in wrong order, making the interpretation of raw information difficult and not straightforward. Users, on the other hand, are interested in a much higher view, the overall state of their job.

For these reasons the raw events undergo complex processing, yielding a high level view, the *job state*, that is the primary type of data presented to the user. Various job states form nodes of the job state diagram (Fig. 1). See Appendix B for a list of the individual states.

L&B defines a *job state machine* that is responsible for updating the job state on receiving a new event. The logic of this algorithm is non-trivial; the rest of this section deals with its main features.

Transitions between the job states happen on receiving events of particular type coming from particular sources. There may be more distinct events assigned to a single edge of the state diagram. For instance, the job becomes *Scheduled* when it enters batch system queue of a Grid computing element. The fact is witnessed by either *Transfer/OK* event reported by the job submission service or by *Accept* event reported by the computing element. Receiving any one of these events (in any order) triggers the state change.

This way, the state machine is highly fault-tolerant—it can cope with delayed, reordered or even lost events. For example, when a job is in the *Waiting* state and the *Done* event arrives, it is not treated

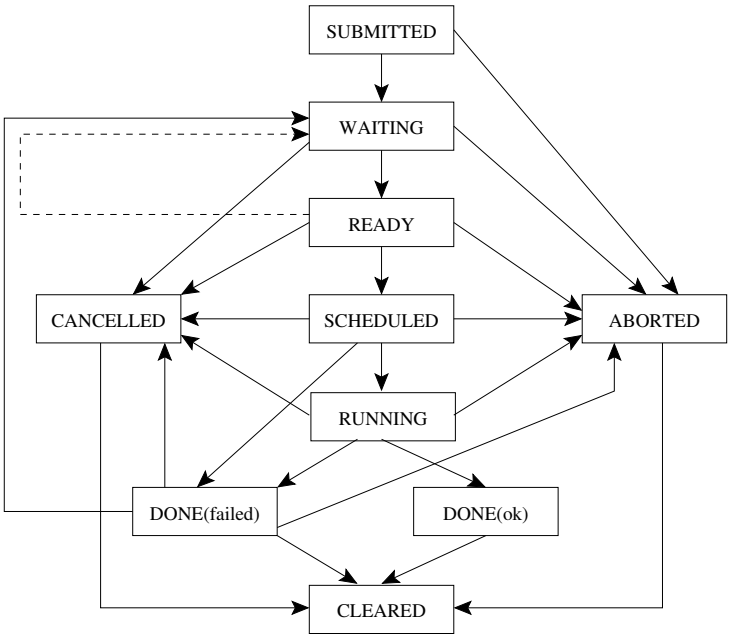

Figure 1: L&B job state diagram

as inconsistency but it is assumed that the intermediate events are delayed or lost and the job state is switched to the *Done* state directly.

The L&B events carry various common and event-type specific attributes, for example *timestamp* (common) or *destination* (*Transfer* type). The job state record contains, besides the major state identification, similar attributes, for example an array of timestamps indicating when the job entered each state, or *location*—identification of the Grid component which is currently handling the job. Updating the job state attributes is also the task of the state machine, employing the above mentioned fault tolerance—despite a delayed event cannot switch the major job state back it still may carry valuable information to update the job state attributes.

Jobs monitored by L&B service may have different type. For gLite jobs, L&B supports simple jobs and jobs representing *set of jobs – DAGs* (with dependencies between subjobs described by direct acyclic graph) and *collections* (without dependencies between subjobs). In these cases, subjobs are monitored in standard way, with one extensions - when job status is changed, information is propagated also to the job representing corresponding collection or DAG. Job representing collection or DAG can be used to monitor status of the set, including information like how many subjobs is already finished etc. Support for non-gLite jobs, namely for PBS or Condor systems, is described in section 1.4.6

1.1.4 EVENT ORDERING

As described above, the ability to correctly order arriving events is essential for the job state computation. As long as the job state diagram was acyclic (which was true for the initial WMS release), each event had its unique place in the expected sequence hence event ordering could always be done implicitly from the context. However, this approach is not applicable once job resubmission yielding cycles in the job state diagram was introduced.

Event ordering that would rely on timestamps assigned to events upon their generation, assuming strict clock synchronization over the Grid, turned to be a naive approach. Clocks on real machines are not

precisely synchronized and there are no reliable ways to enforce synchronization across administrative domains.

To demonstrate a problem with desynchronized clocks, that may lead to wrong event interpretation, let us consider a simplified example in Tab. 1. We assume that the workload manager (WM) sends the job to

1.	WM: Accept	6.	WM: Accept
2.	WM: Match A	7.	WM: Match B
3.	WM: Transfer to A	8.	WM: Transfer to B
4.	CE A: Accept	9.	CE B: Accept
5.	CE A: Run	10.	CE B: Run
...	A dies		

Table 1: Simplified L&B events in the CE failure scenario

a computing element (CE) A, where it starts running but the job dies in the middle. The failure is detected and the job is resubmitted back to the WM which sends it to CE B then. However, if A's clock is ahead in time and B's clock is correct (which means behind the A's clock), the events in the right column are treated as delayed. The state machine will interpret events incorrectly, assuming the job has been run on B before sending it to A. The job would always (assuming the A's events arrive before B's events to the L&B) be reported as "*Running* at A" despite the real state should follow the *Waiting* ... *Running* sequence. Even the *Done* event can be sent by B with a timestamp that says this happened before the job has been submitted to A and the job state will end with a discrepancy—it has been reported to finish on B while still reported to run on A.

Therefore we are looking for a more robust and general solution. We can discover severe clock bias if the timestamp on an event is in a future with respect to the time on an L&B server, but this is generally a dangerous approach (the L&B server clock could be severely behind the real time). We decided not to rely on absolute time as reported by timestamps, but to introduce a kind of *logical time* that is associated with the logic of event generation. The principal idea is arranging the pass through the job state diagram (corresponding to a particular job life), that may include loops, into an execution tree that represents the job history. Closing a loop in the pass through the state diagram corresponds to forking a branch in the execution tree. The scenario in Tab. 1 is mapped to the tree in Fig. 2. The approach is quite general—any finite pass through any state diagram (finite directed graph) can be encoded in this way.

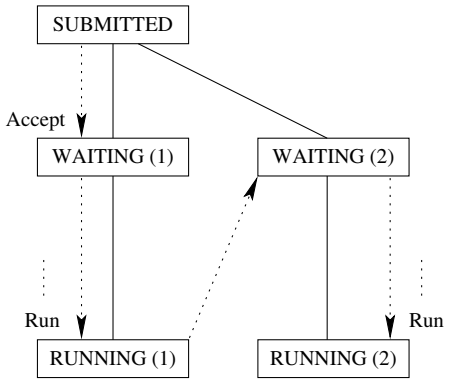

Figure 2: Job state sequence in the CE failure scenario, arranged into a tree. Solid lines form the tree, arrows show state transitions.

Our goal is augmenting L&B events with sufficient information that

- identifies uniquely a branch on the execution tree,
- determines the sequence of events on the branch,
- orders the branches themselves, which means that it determines which one is more recent.

If such information is available, the execution tree can be reconstructed on the fly as the events arrive, and even delayed events are sorted into the tree correctly. An incoming event is considered for job state computation only if it belongs to the most recent branch.

The situation becomes even more complicated when the *shallow resubmission* WM advanced feature is enabled. In this mode WM may resubmit the job before being sure the previous attempt is really unsuccessful, potentially creating multiple parallel instances of the job. The situation maps to several branches of the execution tree that evolve really in parallel. However, only one of the job instances becomes active (really running) finally; the others are aborted. Because the choice of active instance is done later, it may not correspond to the most recent execution branch. Therefore, when an event indicating the choice of active instance arrives, the job state must be recomputed, using the corresponding active branch instead the most recent one.

Section 1.2.5 describes the current implementation of event ordering mechanism based on ideas presented here.

1.1.5 QUERIES AND NOTIFICATIONS

According to the GMA classification the user retrieves data from the infrastructure in two modes, called *queries* and *notifications* in L&B.

Querying L&B is fairly straightforward—the user specifies query conditions, connects to the querying infrastructure endpoint, and receives the results. For "single job" queries, where jobid is known, the endpoint (the appropriate L&B server) is inhered from the jobid. More general queries must specify the L&B server explicitely, and their semantics is intentionally restricted to "all such jobs known here". We trade off generality for performance and reliability, leaving the problem of finding the right query endpoint(s), the right L&B servers, to higher level information and service-discovery services.

If the user is interested in one or more jobs, frequent polling of the L&B server may be cumbersome for the user and creates unnecessary overload on the sever. A notification subscription is therefore available, allowing users to subscribe to receive notification whenever a job starts matching user specified conditions. Every subscription contains also the location of the user's listener; successful subscription returns time-limited *notification handle*. During the validity period of the subscription, the L&B infrastructure is responsible for queuing and reliable delivery of the notifications. The user may even re-subscribe (providing the original handle) with different listener location (for example moving from office to home), and L&B re-routes the notifications generated in the meantime to the new destination. The L&B event delivery infrastructure is reused for the notification transport. Alongside it, there is a possibility to deliver notification messages thourgh the messaging infratstructure.

1.1.6 LOCAL VIEWS

WMS components are, besides logging information into L&B, interested in querying this information back in order to avoid the need of keeping per-job state information. However, despite the required information is present in L&B, the standard mode of L&B operation is not suitable for this purpose due to the following reasons:

- Query interface is provided on the L&B component which gathers events belonging to the same job but coming from different sources. Typically, this is a remote service with respect to the event sources (WMS components). Therefore the query operation is sensitive to any network failure that may occur, blocking the operation of the querying service for indefinite time.
- Due to the asynchronous logging semantics, there is a non-zero time window between successful completion of the logging call and the point in time when the logged event starts affecting the query result. This semantics may yield unexpected, seemingly inconsistent outcome.

The problem can be overcome by introducing *local view* on job data. Besides forwarding events to the server where events belonging to a job are gathered from multiple sources, L&B infrastructure can store the logged events temporarily on the event source, and perform the processing described in Sect. 1.1.3 In this setup, the logging vs. query semantics can be synchronous—it is guaranteed that a successfully logged event is reflected in the result of an immediately following query, because no network operations are involved. Only events coming from this particular physical node (but potentially from all services running there) are considered, thus the locality of the view. On the other hand, certain L&B events are designed to contain redundant information, therefore the local view on processed data (job state) becomes virtually complete on a reasonably rich L&B data source like the Resource Broker node.

1.2 CURRENT L&B IMPLEMENTATION

The principal components of the L&B service and their interactions are shown in Figures 3 (gathering and transferring L&B events) and 4 (L&B query and notification services).

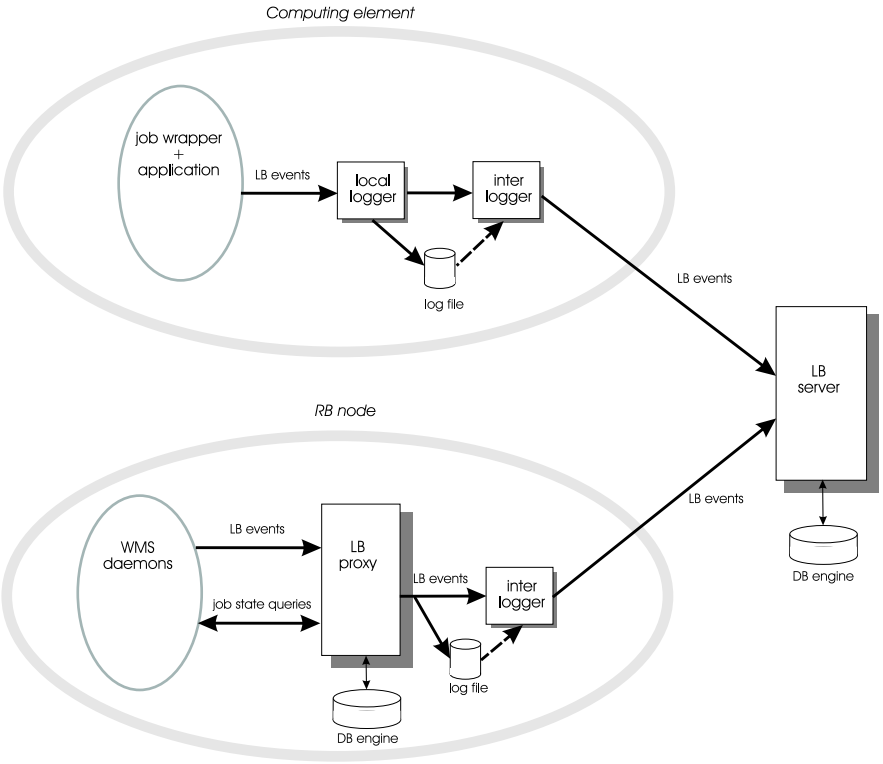

Figure 3: L&B components involved in gathering and transferring the events

1.2.1 L&B API AND LIBRARY

Both logging events and querying the service are implemented via calls to a public L&B API. The complete API (both logging and queries) is available in ANSI C binding, most of the querying capabilities also in C++. These APIs are provided as sets of C/C++ header files and shared libraries. The library implements communication protocol with other L&B components (logger and server), including encryption, authentication etc. Since **L&B version 2.0** an experimental Java binding of the logging API is available.

We do not describe the API here in detail; it is documented in L&B Developer's Guide[3], including complete reference and both simple and complex usage examples.

Events can be also logged with a standalone program (using the C API in turn), intended for usage in scripts.

The query interface is also available as a web-service provided by the L&B server (Sect. 1.2.3).

Finally, certain frequent queries (all user's jobs, single job status, ...) are available as HTML pages (by pointing ordinary web browser to the L&B server endpoint), or as simple text queries (since **L&B version 2.0**) intended for scripts. See 2.5 for details.

1.2.2 LOGGER

The task of the *logger* component is taking over the events from the logging library, storing them reliably, and forwarding to the destination server. The component should be deployed very close to each source of events—on the same machine ideally, or, in the case of computing elements with many worker nodes, on the head node of the cluster[5].

Technically the functionality is realized with two daemons:

- *Local-logger* accepts incoming events, appends them in a plain disk file (one file per Grid job), and forwards to inter-logger. It is kept as simple as possible in order to achieve maximal reliability.
- *Inter-logger* accepts the events from the local-logger, implements the event routing (currently trivial as the destination address is a part of the jobid), and manages delivery queues (one per destination server). It is also responsible for crash recovery—on startup, the queues are populated with undelivered events read from the local-logger files. Finally, the inter-logger purges the files when the events are delivered to their final destination.

1.2.3 SERVER

L&B server is the destination component where the events are delivered, stored and processed to be made available for user queries. The server storage backend is implemented using MySQL database.

Incoming events are parsed, checked for correctness, authorized (only the job owner can store events belonging to a particular job), and stored into the database. In addition, the current state of the job is retrieved from the database, the event is fed into the state machine (Sect. 1.1.3), and the job state updated accordingly.

On the other hand, the server exposes querying interface (Fig. 4, Sect. 1.1.5). The incoming user queries are transformed into SQL queries on the underlying database engine. The query result is filtered, authorization rules applied, and the result sent back to the user.

[5] In this setup logger also serves as an application proxy, overcoming networking issues like private address space of the worker nodes, blocked outbound connectivity etc.

While using the SQL database, its full query power is not made available to end users. In order to avoid either intentional or unintentional denial-of-service attacks, the queries are restricted in such a way that the transformed SQL query must hit a highly selective index on the database. Otherwise the query is refused, as full database scan would yield unacceptable load. The set of indices is configurable, and it may involve both L&B system attributes (for example job owner, computing element, timestamps of entering particular state, . . .) and user defined ones.

The server also maintains the active notification handles(Sect. 1.1.5) , providing the subscription interface to the user. Whenever an event arrives and the updated job state is computed, it is matched against the active handles[6]. Each match generates a notification message, an extended L&B event containing the job state data, notification handle, and the current user's listener location. The event is passed to the *notification inter-logger* via persistent disk file and directly (see Fig. 4). The daemon delivers events either in a standard way, using the specified listener as destination, or forwards them to a messaging broker for delivery through the messaging infrastructure. When using the standard delivery mechanism, the server generates control messages when the user re-subscribes, changing the listener location. Inter-logger recognizes these messages, and changes the routing of all pending events belonging to this handle accordingly.

1.2.4 PROXY

L&B proxy is the implementation of the concept of local view on job state (see Sect. 1.1.6). Since **L&B version 2.0**, L&B proxy is intergrated into L&B server executable. When deployed (on the WMS node in the current gLite middleware) it takes over the role of the local-logger daemon—it accepts the incoming events, stores them in files, and forwards them to the inter-logger.

In addition, the proxy provides the basic principal functionality of L&B server, that is processing events into job state and providing a query interface, with the following differences:

- only events coming from sources on this node are considered; hence the job state may be incomplete,
- proxy is accessed through local UNIX-domain socket instead of network interface,
- no authorization checks are performed—proxy is intended for privileged access only (enforced by the file permissions on the socket),
- aggressive purge strategy is applied—whenever a job reaches a known terminal state (which means that no further events are expected), it is purged from the local database immediately,
- no index checks are applied—we both trust the privileged parties and do not expect the database to grow due to the purge strategy.

1.2.5 SEQUENCE CODES FOR EVENT ORDERING

As discussed in Sect. 1.1.4, sequence codes are used as logical timestamps to ensure proper event ordering on the L&B server. The sequence code counter is incremented whenever an event is logged and the sequence code must be passed between individual Grid components together with the job control. However, a single valued counter is not sufficient to support detection of branch forks within the execution

[6]The current implementation enforces specifying an actual jobid in the subscription hence the matching has minimal performance impact.

tree. When considering again the Computing Element failure scenario described in Sect. 1.1.4, there is no way to know that the counter value of the last event logged by the failed CE A is 5 (see Table 1 on page 11).

Therefore we define a hierarchical *sequence code*—an array of counters, each corresponding to a single Grid component class handling the job[7]. Table 2 below shows the same scenario with a simplified two-counter sequence code. The counters correspond to the WM and CE component classes and they are incremented when each of the components logs an event. When WM receives the job back for resubmission, the CE counter becomes irrelevant (as the job control is on WM now), and the WM counter is incremented again.

1:x	WM: Accept	4:x	WM: Accept
2:x	WM: Match A	5:x	WM: Match B
3:x	WM: Transfer to A	6:x	WM: Transfer to B
3:1	CE A: Accept	6:1	CE B: Accept
3:2	CE A: Run	6:2	CE B: Run
...	A dies		

Table 2: The same CE failure scenario: hierarchical sequence codes. "x" denotes an undefined and unused value.

The state machine keeps the current (highest seen) code for the job, being able to detect a delayed event by simple lexicographic comparison of the sequence codes. Delayed events are not used for major state computation, then. Using another two assumptions (that are true for the current implementation):

- events coming from a single component arrive in order,
- the only branching point is WM,

it is safe to qualify events with lower WM counter (than the already received one) to belong to inactive branches, hence ignore them even for update of job state attributes.

1.2.6 L&B DATA PROTECTION

Events passed between the L&B components as well as the results of their processing provide detailed information about the corresponding job and its life and users obviously expect the job data provided by the L&B server to be credible, reflecting the real jobs' operation on the Grid. Therefore, the data must be based solely on authentic information generated by legitimate grid components. The job data also provides information about user's activities, which many users want to keep private. In order to provide a sufficient level of security, the L&B infrastructure implements a security mechanism that provides data protection and access control to the data.

All the L&B components communicate solely over secure channels whenever they send data over a network. The TLS protocol [6] is used for both mutual authentication of the client and server and also encryption of the communication. All the L&B components as well as the clients must possess a digital certificate that they use to prove their identity. The L&B infrastructure supports both standard X.509 certificates or proxy certificates [7] that are standard authentication mechanism in the gLite environment. Depending on

[7] Currently the following gLite components: Network Server, Workload Manager, Job Controller, Log Monitor, Job Wrapper, and the application itself.

the server configuration and action requested, the users may be required to present VOMS attributes in their proxy certificates.

By default, access to information about a job is only allowed to the user who submitted the job (that is the job owner). The job owner can also assign an access control list to his or job in the L&B specifying other users who are allowed to read the data from L&B. The ACLs are represented in the GridSite GACL format [8] and are stored in the L&B database along with the job information. The stored ACL are checked on each query requesting the data. The ACLs are under control of the job owner, who can add and remove entries in the ACL arbitrarily using the L&B API or command-line tools (see 2.3.2). Each entry of an ACL can specify either a user subject name, a name of a VOMS group, or an attribute specified in the Full qualified attribute name format (the FQAN support is available since **L&B version 2.0**). An ACL assigned to a job is returned as part of job status information.

Besides of using the ACLs, the L&B administrator can also specify a set of privileged users with access to all job records on a particular L&B server (*super-users*). These privileged users can for example collect information on usage and produce monitoring data based on the L&B information.

1.3 USER INTERACTION

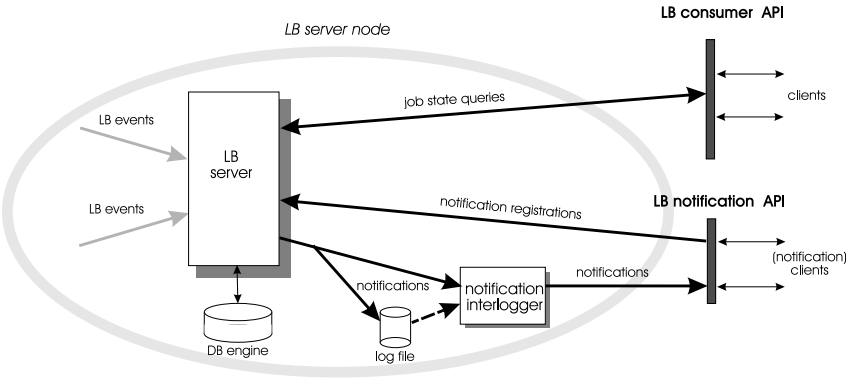

Figure 4: L&B queries and notifications

So far we focused on the L&B internals and the interaction between its components. In this section we describe the interaction of users with the service.

1.3.1 EVENT SUBMISSION

The event submission is mostly implicit, that is it is done transparently by the Grid middleware components on behalf of the user. Typically, whenever an important point in the job life is reached, the involved middleware component logs an appropriate L&B event. This process is not directly visible to the user.

A specific case is the initial registration of the job. This must be done synchronously, as otherwise subsequent events logged for the same job may be refused with a "no such job" error report. Therefore submission of a job to the WMS is the only synchronous event logging that does not return until the job is successfully registered with the L&B server. Moreover, the initial registration is sent by the L&B client library in parallel to L&B proxy (Sect. 1.2.4) and L&B server.

On the other hand, even the registration event may carry large data (e.g. JDL of huge job collection). Therefore also an additional asychronous variant is used, causing two job registration events appear typically.

However, the user may also store information into the L&B explicitly by logging user events—*tags* (or annotations) of the form "name = value". Authorization information is also manipulated in this way.

Description of tools for event submission and ACL manipulation can be found in Section 2

1.3.2 QUERYING INFORMATION

From the user point of view, the information retrieval (*query*) is the most important interaction with the L&B service.

The typical L&B usage are queries on the high-level job state information. L&B supports not only single job queries, it is also possible to retrieve information about jobs matching a specific condition. The conditions may refer to both the L&B system attributes and the user annotations. Rather complex query semantics can be supported, for example *Which of my jobs annotated as "apple" or "pear" are already scheduled for execution and are heading to the "garden" computing element?* The L&B Developer's Guide[3] provides a series of similar examples of complex queries.

As another option, the user may retrieve raw L&B events. Such queries are mostly used for debugging, identification of repeating problems, and similar purposes. The query construction refers to event attributes rather than job state.

The query language supports common comparison operators, and it allows two-level nesting of conditions (logically *and*-ed and *or*-ed). Our experience shows that it is sufficiently strong to cover most user requirements while being simple enough to keep the query cost reasonable. Complete reference of the query language can be found in L&B Developer's Guide [3].

1.3.3 NOTIFICATIONS

L&B notifications are the other mode of user interaction.

The L&B infrastructure can notify its users when something interesting happens on an L&B server (typically a job status change). This allows users to wait comfortably until they are informed by the server, rather than having to poll the L&B server periodically to detect changes.

Users register for notifications via the notification client `glite-lb-notify`, described in Section 2.4 Conditions under which the notifications are sent can be specified. For example – job XY reaches status DONE. In **L&B version 1.x**, one or more JOBID's are required in the condition and only a single occurence of a specific attribute is allowed among ANDed conditions. More complex conditions are allowed since **L&B version 2.0**

The registration is then delivered to the L&B server where it is stored. At the same time, the server generates a unique notification ID for the registration and returns it to the user.

The registration exists only for a limited amount of time. Validity information is returned by L&B server together with the notification ID when registering. During this period the user can attach to the server and receive notification messages, change conditions which triger the notification, prolong validity of the registration, or remove the registration from the L&B server. It is also possible to specify on registration that notification messages for a given registration are not supposed to be delivered through the notification client but through the messaging infrastructure instead.

While the registration is valid, the user is able to repeatably connect to the L&B server from different places in the network and continue receiving notifications associated with the given notification ID. When relying on L&B's event delivery infrastructure to deliver messages, even notifications generated while the user was not connected are stored and waiting until the user reconnects.

1.3.4 CAVEATS

L&B is designed to perform well in the unreliable distributed Grid environment. An unwelcome but inevitable consequence of this design are certain contra-intuitive features in the system behavior, namely:

- Asynchronous, possibly delayed event delivery may yield seemingly inconsistent view on the job state with respect to information that is available to the user via different channels. For example the user may know that her job terminated because of monitoring the application progress directly, but the L&B *Done* events indicating job termination are delayed so that L&B reports the job to be still in the *Running* state.
- Due to the reasons described in Sect. 1.1.4 L&B is rather sensitive to event ordering based on sequence codes. The situation becomes particularly complicated when there are multiple branches of job execution. Consequently the user may see an L&B event that is easily interpreted that it should switch the job state, however, it has no effect in fact because of being (correctly) sorted to an already inactive branch.
- L&B is not a permanent job data storage. The data get purged from the server on timeout, unrelated to any user's action. Therefore, the L&B query may return "Identifier removed" error message (or not include the job in a list of retrieved jobs) even if the same previous L&B query behaved differently.

1.4 ADVANCED USE

The usability of the L&B service is not limited to the simple tasks described earlier. It can be easily extended to support real-time job monitoring (not only the notifications) and the aggregate information collected in the L&B servers is a valuable source of data used for post-mortem statistical analysis of jobs and also the Grid infrastructure behavior. Moreover, L&B data can be used to improve scheduling decisions.

1.4.1 L&B AND REAL TIME MONITORING

The L&B server is extended to provide quickly and without any substantial load on the database engine the following data:

1. number of jobs in the system grouped by internal status (*Submitted*, *Running*, *Done*, ...),
2. number of jobs that reached final state in the last hour,
3. associated statistics like average, maximum, and minimum time spent by jobs in the system,
4. number of jobs that entered the WMS system in the last hour.

L&B server can be regularly queried to provide this data to give an overview about both jobs running on the Grid and also the behavior of the Grid infrastructure as seen from the job (or end user) perspective. Thus L&B becomes a data source for various real-tim Grid monitoring tools.

1.4.2 R-GMA FEED

*Note: This feature is now obsolete and only available in **L&B version 1.x**.*

The L&B server also supports streaming the most important data—the job state changes—to another monitoring system. It works as the notification service, sending information about job state changes to a specific listener that is the interface to a monitoring interface. As a particular example of such a generic service, the R-GMA feed component has been developed. It supports sending job state changes to the R-GMA infrastructure that is part of the Grid monitoring infrastructure used in the EGEE Grid.

Only basic information about job state changes is provided this way, taking into account the security limitation of the R-GMA.

1.4.3 L&B JOB STATISTICS

Data collected within the L&B servers are regularly purged, complicating thus any long term post-mortem statistical analysis. Without a Job Provenance, the data from the L&B must be copied in a controlled way and made available in an environment where even non-indexed queries can be asked.

Using the L&B Job Statistics tools, one dump file per job is created when the job reaches a terminal state. These dump files can be further processed to provide and XML encoded Job History Record[8] that contains all the relevant information from the job life. The Job History Records are fed into a statistical tools to reveal interesting information about the job behavior within the Grid.

This functionality is being replaced by the direct download of all the relevant data from the Job Provenance.

1.4.4 COMPUTING ELEMENT REPUTABILITY RANK

Production operation of the EGEE middleware showed that misbehaving computing elements may have significant impact on the overall Grid performance. The most serious problem is the "black hole" effect—a CE that accepts jobs at a high rate but they all fail there. Such CE usually appears to be free in Grid information services so the resource brokers keep to assign further jobs to it.

L&B data contain sufficient information to identify similar problems. By processing the incoming data the information was made available as on-line auxiliary statistics like rate of incoming jobs per CE, rate of job failure, average duration of job etc. The implementation is lightweight, allowing very high query rate. On the RB the statistics are available as ClassAd functions, allowing the user to specify that similarly misbehaving CE's should be penalized or completely avoided when RB decides where jobs get submitted.

1.4.5 CREAM JOBS

L&B 2.1 implements basic support for CREAM jobs. L&B gathers events from CREAM, performing both the mapping of CREAM attributes to WMS attributes (if possible) and storing CREAM-specific atributes. Thus, the jobs submitted directly to computing elements can be tracked by L&B as well as extended data from WMS. The CREAM job states are mapped to L&B ones according to Appendix B.1

CREAM events are generated by thee CREAM executor and LRMS, the generic *CREAMStatus* event is generated when CREAM notifies that the job status has been changed.

[8] http://egee.cesnet.cz/en/Schema/LB/JobRecord

1.4.6 NON-GLITE EVENT SOURCES

L&B has been enhanced to support also non-gLite events, namely events from PBS or Condor batch systems [9]. These events are handeled differently from gLite events, for a complete list of the PBS and Condor events see Appendix A. Since job states in the batch system slightly differ from the states of a job defined in L&B (see also Appendix B), events are processed separately from gLite events. Both PBS and Condor events has its own state machine that processes the events and determines the now state of the job.

Recently, there were also attempts to use L&B system to transport different types of events: Certificate Revocation Lists or syslog messages. For a detailed description see [10].

1.4.7 CONTROLLING ACCESS TO JOB INFORMATION

Access to information about a job subjects to strong access control mechanism. By default, only the job owner is allowed to access the information about their jobs. There are, however, means how additional rights can be granted to other persons. The L&B server administrator can specify a server-level policy, which grants specific rights to all jobs stored in the server, please refer to the L&B Administrators' guide for more information.

Besides the server-wide configuration, a job owner can also grant access to their jobs to other users. Each job can be assigned an Access control list (ACL), which specifyies which users are allowed to work with the job information. If a job owner wants other users to be allowed to obtain information about their job, they provide an ACL rule granting `READ` access for the users. The users can be identified either by their X.509 subject name or their VOMS attributes. As of **L&B version 3.0**, it is also possible to grant a `TAG` right, which allows multiple users to add user tags to the same job. The current ACL of a job is returned as a part of the job status. Management of the ACL entries is done using logging a special L&B event in a standard way (see 2.3.2 for particular examples).

Starting from **L&B version 3.0**, it is also possible to specify the owner of the actual payload of a job. The feature has been introduced for better support of multi-user pilot jobs, where the pilot submitter differs from the owner of actual payload. In order to set the payload owner, the job owner has to log a `GrantPayloadOwnership` event which specifies the subject owner of the payload owner. This event is supposed to come from a pilot job factory which monitors the pilot jobs and keeps an overview about when a particular payload is started. The new payload owner has to confirm the transition with a `TakePayloadOwnership` logged to the job using his or her credentials. The payload and job owners have the same rights to the jobs, they both can query the job, etc. The identity of the current payload owner is returned as part of the job status information.

2 USER TOOLS

In this section we give a description of the CLI tools that a regular grid user might want to use. If not stated otherwise, the tools are distributed in the `glite-lb-client` package.

2.1 ENVIRONMENT VARIABLES

Behaviour of the commands can be changed by setting enviroment variables, specifing mostly location of servers or setting timeouts for various operations. For a complete list of environment variables, their form and default value description, see Appendix C. Setting the environment variable is for some commands mandatory, so reading the documentaion below and appropriate manpages is highly recommended.

2.2 GLITE-WMS-JOB-STATUS AND GLITE-WMS-JOB-LOGGING-INFO

We start with tools that are distributed in `glite-wms-ui-cli-python` package and that can be found probably on all UI machines. Description of all user commands that are used during the job submission process (generating proxy, creating a JDL describing the job, submitting a job, retrieving output, cancelling a job, etc.) is beoynd this document and it can be found for example in [11]. We mention here only the commands that are related to job monitoring.

Once job has been submitted to WMS (by `glite-wms-job-submit`), a user can retrieve the job status by

```
glite-wms-job-status <jobId>
```

or if job ID is saved in the file

```
glite-wms-job-status -i <job_id_file>
```

or if user wants to see status of all his/her jobs

```
glite-wms-job-status --all
```

List of all possible job states is summarised in Appendix B.

Logging details on submitted job can be accessed via

```
glite-wms-job-logging-info -v <verbosity_level> <job_ID>
```

or if job ID is saved in the file

```
glite-wms-job-logging-info -v <verbosity_level> -i <job_id_file>
```

where verbosity level can be from 0 to 3.

2.3 GLITE-LB-LOGEVENT

Besides the API's L&B offers its users a simple command-line interface for logging events. The command `glite-lb-logevent` is used for this purpose. However, it is intended for internal WMS debugging tests in the first place and should not be used for common event logging because of possibility of confusing L&B server job state automaton.

The command `glite-lb-logevent` is a complex logging tool and the complete list of parameters can be obtained using the `-h` option. However, the only legal user usage is for logging `UserTag` and `ChangeACL` events. The following description is therefore concentrating only on options dealing with these two events.

Command usage is:

```
glite-lb-logevent [-h] [-p] [-c seq_code]
    -j <dg_jobid> -s Application -e <event_name> [key=value ...]
```

where

`-p --priority`	send a priority event
`-c --sequence`	event sequence code
`-j --jobid`	JobId
`-e --event`	select event type (see -e help)

Each event specified after `-e` option has different sub-options enabling to set event specific values.

Address of local-logger, daemon responsible for further message delivery, must be specified by environment variable `GLITE_WMS_LOG_DESTINATION` in a form `address:port`.

Because user is allowed to change ACL or add user tags only for her jobs, paths to valid X509 user credentials has to be set to authorise her. This is done using standard X509 environment variables `X509_USER_KEY` and `X509_USER_CERT`.

For additional information see also manual page glite-lb-logevent(1).

2.3.1 EXAMPLE: LOGGING A USERTAG EVENT

User tag is an arbitrary "name=value" pair with which the user can assign additional information to a job. Further on, LB can be queried based also on values of user tags. L&B treats all values as strings only, semantic meaning is left to user application. For internal reasons, all tag names are stored in lower-case format. Support for case-sensitivenes is planned in future versions of L&B.

In order to add user tag for a job a special event `UserTag` is used. This event can be logged by the job owner using the glite-lb-logevent command (see also sec.2.3). Here we suppose the command is used from user's running application because a correct setting of environment variables needed by the command is assured.

General template for adding user tag is as follows:

```
glite-lb-logevent -s Application -e UserTag
        -j <job_id>
        -c <seq_code>
        --name <tag_name>
        --value <tag_value>
```

where

`<tag_name>` specifies the name of user tag

`<tag_value>` specifies the value of user tag

The user application is always executed from within a JobWrapper script (part of Workload Management System [5]). The wrapper sets the appropriate `JobId` in the environment variable `GLITE_WMS_JOBID`. The user should pass this value to the `-j` option of `glite-lb-logevent`. Similarly, the wrapper sets an initial value of the event sequence code in the environment variable `GLITE_WMS_SEQUENCE_CODE`.

If the user application calls `glite-lb-logevent` just once, it is sufficient to pass this value to the `-c` option. However, if there are more subsequent calls, the user is responsible for capturing an updated sequence code from the stdout of `glite-lb-logevent` and using it in subsequent calls. The L&B design requires the sequence codes in order to be able to sort events correctly while not relying on strictly synchronized clocks.

The example bellow is a job consisting of 100 phases. A user tag phase is used to log the phase currently being executed. Subsequently, the user may monitor execution of the job phases as a part of the job status returned by L&B.

```
#!/bin/sh

for p in `seq 1 100`; do

# log the UserTag event
GLITE_WMS_SEQUENCE_CODE=`glite-lb-logevent -s Application
  -e UserTag
  -j $GLITE_WMS_JOBID -c $GLITE_WMS_SEQUENCE_CODE
  --name=phase --value=$p`

# do the actual computation here
done
```

2.3.2 EXAMPLE: CHANGING JOB ACCESS CONTROL LIST

In order to change the Access Control List (ACL) for a job (see also 1.4.7), a special event `ChangeACL` is used. This event can be logged by the job owner using the `glite-lb-logevent` command (see also Sect. 2.3). The general template for changing the ACL is as follows:

```
glite-lb-logevent -e ChangeACL -s UserInterface -p -j <job_id>
        --user_id <user_id>
        --user_id_type <user_id_type>
        --permission READ
        --permission_type <permission_type> --operation <operation>
```

where

`<job_id>`	specifies the job to change access to
`<user_id>`	specifies the user to grant or revoke permission. The parameter can be either an X.500 name (subject name), a VOMS group (of the form VO:Group), or a Full qualified attribute name (FQAN).
`<user_id_type>`	indicates the type of the user_id given above. `DN`, `GROUP`, and `FQAN` can be given to specify X.500 name, VOMS group, or FQAN, respectively
`<permission>`	ACL permission to change, currently only `READ` is supported. Starting from **L&B version 3.0**, the permission `TAG` can also be used.
`<permission_type>`	Type of permission requested. `ALLOW` or `DENY` can be specified.
`<operation>`	Operation requested to be performed with ACL. `ADD` or `REMOVE` can be specified.

Adding a user specified by his or her subject name to the ACL (that is granting access rights to another user):

```
glite-lb-logevent -e ChangeACL -s UserInterface -p -j <job_id>          \
        --user_id '/O=CESNET/O=Masaryk University/CN=Daniel Kouril'     \
        --user_id_type DN --permission READ --permission_type ALLOW     \
        --operation ADD
```

Removing a user specified by his or her subject name from the ACL (that is revoking access right to another user):

```
glite-lb-logevent -e ChangeACL -s UserInterface -p -j <job_id>          \
        --user_id '/O=CESNET/O=Masaryk University/CN=Daniel Kouril'     \
        --user_id_type DN --permission READ --permission_type ALLOW     \
        --operation REMOVE
```

Adding a VOMS attribute to the ACL:

```
glite-lb-logevent -e ChangeACL -s UserInterface -p -j <job_id>          \
        --user_id '/VOCE/Role=Administrator' --user_id_type FQAN        \
        --permission TAG --permission_type ALLOW                       \
        --operation ADD
```

Note that **L&B version 1.x** supported only using VOMS group names, not full FQANs, whose support has been introduced in **L&B version 2.0**. **L&B version 1.x** also did not allow the users to use symbolic names for the values specifying ACL setting and integers must be used instead. For example, to grant access right on a **L&B version 1.x** server one has to use following syntax:

```
glite-lb-logevent -e ChangeACL -s UserInterface -p -j <job_id>          \
        --user_id '/O=CESNET/O=Masaryk University/CN=Daniel Kouril'     \
        --user_id_type 0 --permission 1 --permission_type 0 --operation 0
```

2.3.3 EXAMPLE: SETTING PAYLOAD OWNER

In order to change the owner of the payload (see also 1.4.7), a pair of L&B events is used. In order for a job owner to specify a new owner of the payload, the `GrantPayloadOwnership` event is used, for example:

```
glite-lb-logevent -e GrantPayloadOwnership -s UserInterface -j <job_id> \
        --payload_owner <subject_name>
```

where

`<job_id>`	specifies the job to change access to
`<subject_name_id>`	specifies the X.509 subject name of the new payload user.

The new payload owner confirm they accept the ownership using the `TakePayloadOwnership` event. Note that this event event must be logged using the credentials of the new user:

```
glite-lb-logevent -e TakePayloadOwnership -s UserInterface -j <job_id>
```

2.4 GLITE-LB-NOTIFY

`glite-lb-notify` is a fairly simple wrapper on the L&B notification API (see also [3]). It allows to create a notification (with a restricted richness of specifying conditions), bind to it for receiving notifications, and drop it finally.

L&B notification is a user-initiated trigger at the server. Whenever a job enters a state matching conditions specified with the notification, the current state of the job is sent to the notification client. On the other hand, the notification client is a network listener which receives server-initiated connections to get the notifications. Unless `-s` is specified, the notification library creates the listener socket.

Within the notification validity, clients can disappear and even migrate. However, only a single active client for a notification is allowed.

L&B server and port to contact must be specified with GLITE_WMS_NOTIF_SERVER environment variable.

`glite-lb-notify` is supported by **L&B version 2.x** and newer. In **L&B version 1.x**, `glite-lb-notify` with very limited functionality can be found in `examples` directory.

`glite-lb-notify` support these actions:

`new`	Create new notification registration.
`bind`	Binds an notification registration to a client.
`refresh`	Enlarge notification registration validity.
`receive`	Binds to an existing notification registration and listen to server.
`drop`	Drop the notification registration.

For action `new`, command usage is:

```
glite-lb-notify new [ { -s socket_fd | -a fake_addr } -t requested_validity
         -j jobid  { -o owner | -O }  -n network_server
         -v virtual_organization --states state1,state2,... -c -f flags]
```

For action `bind`, command usage is:

```
glite-lb-notify bind [ { -s socket_fd | -a fake_addr } -t requested_validity ]
         notifid
```

For action `refresh`, command usage is:

```
glite-lb-notify refresh [-t requested_validity ] notifid
```

For action `receive`, command usage is:

```
glite-lb-notify receive  [ { -s socket_fd | -a fake_addr } ] [-t requested_validity ] [-i ti
```

For action `drop`, command usage is:

```
glite-lb-notify drop notifid
```

where

`notifid`	Notification ID
`-s socket_fd`	allows to pass a opened listening socket
`-a fake_addr`	fake the client address
`-t requested_validity`	requested validity of the notification (in seconds)
`-j jobid`	job ID to connect notification registration with
`-o owner`	match this owner DN
`-O`	match owner on current user's DN
`-n network_server`	match only this network server (WMS entry point)
`-v virtual_organization`	match only jobs of this Virtual Organization
`-i timeout`	timeout to receive operation in seconds
`-f field1,field2,...`	list of status fields to print (only owner by default)
`-c`	notify only on job state change
`-S, -state state1,state2,...`	match on events resulting in listed states
`-r`	refresh automatically the notification registration while receiving data

For additional information see also manual page glite-lb-notify(1).

2.4.1 EXAMPLE: REGISTRATION AND WAITING FOR SIMPLE NOTIFICATION

Following steps describe basic testing procedure of the notification system by registering a notification on any state change of this job and waiting for notification.

Register notification for a given jobid with validity 2 hours (7200 seconds):

```
GLITE_WMS_NOTIF_SERVER=skurut68-2.cesnet.cz:9100 glite-lb-notify new \
 -j https://skurut68-2.cesnet.cz:9100/D1qbFGwvXLnd927JOcja1Q -t 7200
```

returns:

```
notification ID: https://skurut68-2.cesnet.cz:9100/NOTIF:tOsgB19Wz-M884anZufyUw
```

Wait for notification (with timeout 120 seconds):

```
GLITE_WMS_NOTIF_SERVER=skurut68-2.cesnet.cz:9100 glite-lb-notify receive \
 -i 120 https://skurut68-2.cesnet.cz:9100/NOTIF:tOsgB19Wz-M884anZufyUw
```

returns:

```
notification is valid until: '2008-07-29 15:04:41' (1217343881)
https://skurut68-2.cesnet.cz:9100/D1qbFGwvXLnd927JOcja1Q        Waiting
    /DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
https://skurut68-2.cesnet.cz:9100/D1qbFGwvXLnd927JOcja1Q        Ready
    /DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
https://skurut68-2.cesnet.cz:9100/D1qbFGwvXLnd927JOcja1Q        Scheduled
    /DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
https://skurut68-2.cesnet.cz:9100/D1qbFGwvXLnd927JOcja1Q        Running
    /DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
```

Destroy notification:

```
GLITE_WMS_NOTIF_SERVER=skurut68-2.cesnet.cz:9100 glite-lb-notify drop \
  https://skurut68-2.cesnet.cz:9100/NOTIF:tOsgB19Wz-M884anZufyUw
```

2.4.2 EXAMPLE: WAITING FOR NOTIFICATIONS ON ALL USER'S JOBS

Instead of waiting for one job, user may want to accept notification about state changes of all his jobs.

Register notification for actual user:

```
GLITE_WMS_NOTIF_SERVER=skurut68-2.cesnet.cz:9100 glite-lb-notify new -O
```

returns:

```
notification ID: https://skurut68-2.cesnet.cz:9100/NOTIF:tOsgB19Wz-M884anZufyUw
```

And continue with `glite-lb-notify receive` similarly to previous example. But in this case, we want to display also other information about job – not job owner, but destination (where job is running) and location (which component is serving job):

```
GLITE_WMS_NOTIF_SERVER=skurut68-2.cesnet.cz:9100 glite-lb-notify receive \
 -i 240 -f destination,location \
 https://skurut68-2.cesnet.cz:9100/NOTIF:tOsgB19Wz-M884anZufyUw
```

returns:

```
notification is valid until: '2008-07-29 15:43:41' (1217346221)

https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Waiting
  (null) NetworkServer/erebor.ics.muni.cz/
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Waiting
  (null)  destination queue/erebor.ics.muni.cz/
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Waiting
  (null) WorkloadManager/erebor.ics.muni.cz/
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Waiting
  (null)  name of the called component/erebor.ics.muni.cz/
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Waiting
```

```
  destination CE/queue WorkloadManager/erebor.ics.muni.cz/
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Waiting
  destination CE/queue WorkloadManager/erebor.ics.muni.cz/
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Ready
  destination CE/queue destination queue/erebor.ics.muni.cz/
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Ready
  destination CE/queue JobController/erebor.ics.muni.cz/
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Ready
  destination CE/queue LRMS/destination hostname/destination instance
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Ready
  destination CE/queue LogMonitor/erebor.ics.muni.cz/
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Scheduled
  destination CE/queue LRMS/destination hostname/destination instance
https://skurut68-2.cesnet.cz:9100/qbRFxDFCg2qO4-9WYBiiig        Running
  destination CE/queue LRMS/worknode/worker node
```

2.4.3 EXAMPLE: REGISTERING FOR NOTIFICATIONS TO BE DELIVERED OVER ACTIVEMQ

Delivering notification messages over the messaging infrastructure provided by ActiveMQ is a feature introduced in **L&B version 3.0**. It uses the fake address capability (`-a` option) to specify messaging topic to use when generating messages.

```
GLITE_WMS_NOTIF_SERVER=skurut68-2.cesnet.cz:9100 glite-lb-notify new \
 -O -a x-msg://grid.emi.lbexample
```

Rather than using the L&B notification API to receive messages, access the messaging infrastructure and tap into the given messaging topic (`grid.emi.lbexample` in our case).

Note that production environments can impose restrictions on topic names. In the context of EGI, for instance, prefix "`grid.emi.`" is mandatory.

2.4.4 EXAMPLE: WAITING FOR MORE NOTIFICATIONS ON ONE SOCKET

The foloving example demonstrates possibility to reuse existing socket for receiving multiple notifications. Perl script `notify.pl` (available in examples directory) creates socket, which is then reused for all `glite-lb-notify` commands.

```
GLITE_WMS_NOTIF_SERVER=skurut68-2.cesnet.cz:9100 NOTIFY_CMD=glite-lb-notify \
 ./notify.pl -O
```

returns:

```
notification ID: https://skurut68-2.cesnet.cz:9100/NOTIF:EO73rjsmexEZJXuSoSZVDg
valid: '2008-07-29 15:14:06' (1217344446)
got connection
https://skurut68-2.cesnet.cz:9100/ANceuj5fXdtaCCkfnhBIXQ        Submitted
```

```
/DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
glite-lb-notify: Connection timed out (read message)
got connection
https://skurut68-2.cesnet.cz:9100/p2jBsy5WkFItY284lW2o8A        Submitted
/DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
glite-lb-notify: Connection timed out (read message)
got connection
https://skurut68-2.cesnet.cz:9100/p2jBsy5WkFItY284lW2o8A        Waiting
/DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
```

2.4.5 EXAMPLE: WAITING FOR NOTIFICATIONS ON JOBS REACHING TERMINAL STATES

This example shows how to set up notifications for jobs reaching state *done* or *aborted*. Since we prefer to receive just one notification per job, we will also use the `-c` option, which makes sure that notifications will be generated only on actual job state change.

```
GLITE_WMS_NOTIF_SERVER=skurut68-2.cesnet.cz:9100 glite-lb-notify new \
 --state done,aborted -c
```

returns:

```
notification ID: https://skurut68-2.cesnet.cz:9100/NOTIF:6krjMRshTouH2n7D9t-xdg
valid: '2009-04-30 06:59:18 UTC' (1241074758)
```

Wait for notification (with timeout 120 seconds):

```
GLITE_WMS_NOTIF_SERVER=skurut68-2.cesnet.cz:9100 glite-lb-notify receive \
 -i 120 https://skurut68-2.cesnet.cz:9100/NOTIF:6krjMRshTouH2n7D9t-xdg
```

returns:

```
https://skurut68-2.cesnet.cz:9100/eIbQNz3oHpv-OkYVu-cXNg Done
    /DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
https://skurut68-2.cesnet.cz:9100/GpBy2gfIZOAXR2hKOAYGgg Aborted
    /DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
https://skurut68-2.cesnet.cz:9100/KWXmsqvsTQKizQ4OMiXItA Done
    /DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
https://skurut68-2.cesnet.cz:9100/O1zs50Nm1r03vx2GLyaxQw Done
    /DC=cz/DC=cesnet-ca/O=Masaryk University/CN=Miroslav Ruda
```

2.5 HTML AND PLAIN TEXT INTERFACE

Since the gLite jobId has the form of a unique URL, it is possible to cut and paste it directly to the web browser to view the information about the job (esp. its status), e.g.

```
firefox https://pelargir.ics.muni.cz:9000/1234567890
```

To list all user's jobs, it is possible to query only the L&B server address, e.g.

```
firefox https://pelargir.ics.muni.cz:9000
```

To list all user's notification registration curently valid on a given L&B server, use a URL constructed as in folowing example:

```
firefox https://pelargir.ics.muni.cz:9000/NOTIF
```

A notification ID also have a form of URL. If you direct your browser to a particular notification ID, the L&B server will provide the notification details for it.

```
firefox https://pelargir.ics.muni.cz:9000/NOTIF:1234567890
```

In all cases it is necessary to have the user certificate installed in the browser.

Since **L&B version 2.0**, it is also possible to obtain the above results in a machine readable `key=value` form by adding a suffix `text` to the above URLs. For example

```
curl -3 --key $X509_USER_KEY --cert $X509_USER_CERT \
  --capath /etc/grid-security/certificates \
  https://pelargir.ics.muni.cz:9000?text
```

or

```
curl -3 --key $X509_USER_KEY --cert $X509_USER_CERT \
  --capath /etc/grid-security/certificates \
  https://pelargir.ics.muni.cz:9000/1234567890?text
```

2.6 JOB STATE CHANGES AS AN RSS FEED

The L&B includes an RSS interface allowing users to keep trace of their jobs in a very simple way using an RSS reader. The parameters of the RSS feeds are predefined, so no configuration is required.

The address of a feed is given by the URL of the L&B server and a */RSS:rss_feed_name* postfix, e.g.

```
https://pelargir.ics.muni.cz:9000/RSS:finished
```

There are currently 3 feeds implemented in LB:

- *finished* for jobs in terminal states (Done/OK, Aborted and Canceled)
- *running* for running jobs
- *aborted* for aborted jobs

2.7 OTHER USEFUL TOOLS

For debugging purposes, low-level commands for getting L&B job status and job related events are provided in `examples` directory (`glite-lb-job_status` and `glite-lb-job_log`). The same directory contains also debugging commands for getting of all user jobs (`glite-lb-user_jobs`) and CE-reputability rank (see Section 1.4.4, `glite-lb-stats`).

3 TROUBLESHOOTING

Please, report usage problems via the GGUS support system at

https://gus.fzk.de/index.html

Apparent software bugs are tracked in Savannah at

https://savannah.cern.ch/bugs/?func=additem&group=jra1mdw

When submitting a L&B spcecific problem/bug, the following information might be useful:

- version of software used (all `glite-lb-*` packages installed at your site)
- description of the problem, the `jobId`, addresses of all relevant machines (L&B server, ...), environment variables set, etc.
- output from the following commands:

  ```
  glite-wms-job-status <jobId>
  glite-wms-job-logging-info -v 3 <jobID>
  ```

- information on your proxy:

  ```
  voms-proxy-info -debug -all
  ```

  or, if VOMS client is not installed,

  ```
  grid-proxy-info -debug
  ```

- sometimes additional information can be found in the output from the commands

  ```
  glite-lb-job_status <jobId>
  glite-lb-job_log <jobId>
  ```

  that should be available in the `$GLITE_LOCATION/examples` directory,

- appropriate excerpts from the logs on the server side are also highly appreciated, please tell your administrator to look at the Troubleshooting section in LB Administrator's Guide [2] to follow the steps there and provide you the necessary information.

Users are encouradged to send developers all non-bugs comments and questions by email to address egee-jra1@lindir.ics.muni.cz.

4 FAQ—FREQUENTLY ASKED QUESTIONS

4.1 JOB IN STATE 'RUNNING' DESPITE HAVING RECEIVED THE 'DONE' EVENT FROM LRMS

Jobs stay in state *Running* until a *Done* event is received from the workload management system. *Done* events from local resource managers are not enough since the job in question may have been resubmitted in the meantime.

4.2 WMS CANNOT PURGE JOBS OR PERFORM OTHER PRIVILEGED TASKS

In short, WMS has not been given adequate permissions when configuring the L&B server. You need to modify your configuration and restart the server:

4.2.1 FOR L&B VERSION 3.0.11 OR HIGHER, USING YAIM

Modify your `siteinfo.def`, specifying the DN of your WMS server in YAIM parameter `GLITE_LB_WMS_DN`; for instance:

```
GLITE_LB_WMS_DN=/DC=cz/DC=cesnet-ca/O=CESNET/CN=wms01.cesnet.cz
```

Then rerun YAIM: `/opt/glite/yaim/bin/yaim -c -s site-info.def -n glite-LB`

This will give your WMS exactly the right permissions to carry out all required operations.

4.2.2 FOR ALL VERSIONS OF L&B, USING YAIM

Modify your `siteinfo.def`, specifying the DN of your WMS server in YAIM parameter `GLITE_LB_SUPER_USERS`; for instance:

```
GLITE_LB_SUPER_USERS=/DC=cz/DC=cesnet-ca/O=CESNET/CN=wms01.cesnet.cz
```

Then rerun YAIM: `/opt/glite/yaim/bin/yaim -c -s site-info.def -n glite-LB`

This will give your WMS adequate rights to perform its operations and requests (running purge, querying for statistics, etc.) but it will also grant it additional administrator rights (such as granting job ownership). On newer installations, the method explained in section 4.2.1 is preferrable.

4.2.3 FOR L&B VERSION 2.1 OR HIGHER, WITHOUT YAIM

L&B's authorization settings can be found in file `[/opt/glite]/etc/glite-lb/glite-lb-authz.conf`

Permit actions `PURGE`, `READ_ALL` and `GET_STATISTICS` for your WMS and restart the L&B server. This will lead to results equivalent to 4.2.1. For instance, change the adequate sections in `glite-lb-authz.conf` to:

```
action "READ_ALL" {
        rule permit {
                subject = "/DC=cz/DC=cesnet-ca/O=CESNET/CN=wms01.cesnet.cz"
```

```
        }
}

action "PURGE" {
        rule permit {
                subject = "/DC=cz/DC=cesnet-ca/O=CESNET/CN=wms01.cesnet.cz"
        }
}

action "GET_STATISTICS" {
        rule permit {
                subject = "/DC=cz/DC=cesnet-ca/O=CESNET/CN=wms01.cesnet.cz"
        }
}
```

APPENDIX

A L&B EVENT TYPES

Complete list of all events' names together with their description follows.

EVENTS FOR GLITE WORLD:

1.	**Transfer:**	Start, success, or failure of job transfer to another component.
2.	**Accepted:**	Accepting job (successful counterpart to Transfer).
3.	**Refused:**	Refusing job (unsuccessful counterpart to Transfer).
4.	**EnQueued:**	The job has been enqueued in an inter-component queue.
5.	**DeQueued:**	The job has been dequeued from an inter-component queue.
6.	**HelperCall:**	Helper component is called.
7.	**HelperReturn:**	Helper component is returning the control.
8.	**Running:**	Job wrapper started.
9.	**Resubmission:**	Result of resubmission decision.
10.	**Done:**	Execution terminated (normally or abnormally).
11.	**Cancel:**	Cancel operation has been attempted on the job.
12.	**Abort:**	Job aborted by system.
13.	**Clear:**	Job cleared, output sandbox removed
14.	**Purge:**	Job is purged from bookkepping server.
15.	**Match:**	Matching CE found.
16.	**Pending:**	No matching CE found yet.
17.	**RegJob:**	New job registration.
18.	**Chkpt:**	Application-specific checkpoint record.
19.	**Listener:**	Listening network port for interactive control.
20.	**CurDescr:**	Current state of job processing (optional event).
21.	**UserTag:**	User tag – arbitrary name=value pair.
22.	**ChangeACL:**	Management of ACL stored on bookkepping server.
23.	**Notification:**	Management of notification service.
24.	**ResourceUsage:**	Resource (CPU, memory etc.) consumption.
25.	**ReallyRunning:**	User payload started.
26.	**Suspend:**	Job execution (queuing) was suspended.
27.	**Resume:**	Job execution (queuing) was resumed.
28.	**CollectionState:**	State of the collection.
29.	**GrantPayloadOwnership:**	Hand over ownership of actual job payload (e.g. of a pilot job)
30.	**TakePayloadOwnership:**	Take over ownership of actual job payload

EVENTS FOR PBS WORLD:

101.	**PBSQueued:**	Job enqued
102.	**PBSMatch:**	Scheduler created exec host
103.	**PBSPending:**	Scheduler is not able to find exec host, or some error occured
104.	**PBSRun:**	Job attempted to be run by the logging component
105.	**PBSRerun:**	Job rerun requested
106.	**PBSDone:**	Job terminated
107.	**PBSDequeued:**	Job dequeued
108.	**PBSResourceUsage:**	Resources requested/consumed
109.	**PBSError:**	Any error occured

EVENTS FOR CONDOR WORLD:

201.	**CondorMatch:**	Job MATCHed
202.	**CondorReject:**	Job REJECTed
203.	**CondorShadowStarted:**	Condor Shadow Started
204.	**CondorShadowExited:**	Condor Shadow Exited
205.	**CondorStarterStarted:**	Condor Starter Started
206.	**CondorStarterExited:**	Condor Starter Exited
207.	**CondorResourceUsage:**	Resources requested/consumed
208.	**CondorError:**	Any Error occured

EVENTS FOR CREAM WORLD:

301.	**CREAMStart:**	Start processing registered job
302.	**CREAMPurge:**	Purge request (by user)
303.	**CREAMAccepted:**	Accepting job (successful counterpart to Transfer).
304.	**CREAMStore:**	
305.	**CREAMCall:**	Processing command and calling BLAH or LRMS
306.	**CREAMRunning:**	
307.	**CREAMReallyRunning:**	
308.	**CREAMDone:**	
309.	**CREAMCancel:**	
310.	**CREAMAbort:**	
311.	**CREAMStatus:**	
312.	**CREAMSuspend:**	Job execution (queuing) was suspended.
313.	**CREAMResume:**	Job execution (queuing) was resumed.

EVENTS FOR TRANSFER WORLD:

401.	**FileTransferRegister:**	register file transfer
402.	**FileTransfer:**	transfer job logs progress
403.	**Sandbox:**	event for logging relationship between (compute) job and (file) transfer job

B L&B JOB STATES

Complete list of all job' states together with their description follows.

Submitted: Entered by the user to the User Interface or registered by Job Partitioner.
Waiting: Accepted by WMS, waiting for resource allocation.
Ready: Matching resources found.
Scheduled: Accepted by LRMS queue.
Running: Executable is running.
Done: Execution finished, output is available.
Cleared: Output transfered back to user and freed.
Aborted: Aborted by system (at any stage).
Cancelled: Cancelled by user.
Unknown: Status cannot be determined.
Purged: Job has been purged from bookkeeping server (for LB-RGMA interface).

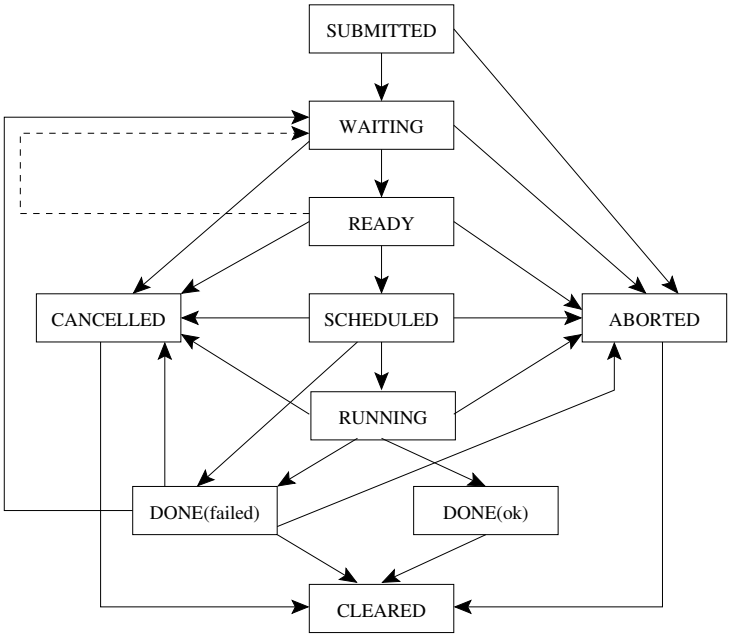

Figure 5: L&B job state diagram

B.1 CREAM JOB STATES MAPPING

Support of CREAM jobs is available since **L&B version 2.1**. This is the implemented mapping of job states between CREAM and L&B:

CREAM state	L&B state
Registered	Submited
Pending	Waiting
Idle	Scheduled
Running	Running
Really Running	Running
Done OK	Done
Done Failed	Done
Aborted	Aborted
Cancelled	Cancelled

C ENVIRONMENT VARIABLES

Complete list of all environment variables affecting LB behaviour follows with their description and default values (if applicable).

GLITE_WMS_LOG_DESTINATION	address of the `glite-lb-logd` daemon (for logging events), in form `hostname:port`, default value is `localhost:9002`
GLITE_WMS_LOG_TIMEOUT	timeout (in seconds) for asynchronous logging, default value is `120` seconds, maximum value is `300` seconds
GLITE_WMS_LOG_SYNC_TIMEOUT	timeout (in seconds) for synchronous logging, default value is `120` seconds, maximum value is `600` seconds
GLITE_WMS_NOTIF_SERVER	address of the `glite-lb-bkserver` daemon (for receiving notifications) in form `hostname:port`, for receiving notifications, there is no default value, mandatory for `glite-lb-notify`
GLITE_WMS_NOTIF_TIMEOUT	timeout (in seconds) for notification registration, default value is `120` seconds, maximum value is `1800` seconds
GLITE_WMS_QUERY_SERVER	address of the `glite-lb-bkserver` daemon (for queries), in form `hostname:port`, there is no default value
GLITE_WMS_QUERY_TIMEOUT	timeout (in seconds) for queries, default value is `120` seconds, maximum value is `1800` seconds
GLITE_WMS_LBPROXY_STORE_SOCK	UNIX socket location for logging to LB Proxy, default value is `/tmp/lb_proxy_store.sock`
GLITE_WMS_LBPROXY_SERVE_SOCK	UNIX socket location for queries to LB Proxy, default value is `/tmp/lb_proxy_serve.sock`
GLITE_WMS_LBPROXY_USER	user credentials (DN) when communicating with LB Proxy, there is no default value
X509_USER_CERT, X509_USER_KEY	location of user credentials (certificate and private key), default values are `/.globus/usercert,key.pem`
GLOBUS_HOSTNAME	hostname to appear as event origin, useful only for debugging, default value is hostname
QUERY_SERVER_OVERRIDE	values defined in QUERY_SERVER will override also values in jobid in queries, useful for debugging only, default value `no`
QUERY_JOBS_LIMIT	maximal size of results for query on jobs, default value is `0` (unlimited)
QUERY_EVENTS_LIMIT	maximal size of results for query on events, default value is `0` (unlimited)
QUERY_RESULTS	specifies behavior of query functions when size limit is reached, value can be `None` (no results are returned), `All` (all results are returned, even if over specified limit), `Limited` (size of results is limited to size specified by QUERY_JOBS_LIMIT or QUERY_EVENTS_LIMIT)
CONNPOOL_SIZE	maximal number of open connections in logging library, for developers only, default value is `50`

For backward compatibility, all `GLITE_WMS_*` variables can be prefixed by `EDG_WL_` instead, for example

EDG_WL_LOG_DESTINATION.

REFERENCES

[1] E. Laure, F. Hemmer, F. Prelz, S. Beco, S. Fisher, M. Livny, L. Guy, M. Barroso, P. Buncic, P. Kunszt, A. Di Meglio, A. Aimar, A. Edlund, D. Groep, F. Pacini, M. Sgaravatto, and O. Mulmo. Middleware for the next generation grid infrastructure. In *Computing in High Energy Physics and Nuclear Physics (CHEP 2004)*, 2004.

[2] A. Křenek et al. L&B Administrator's Guide. http://egee.cesnet.cz/en/JRA1/LB/.

[3] A. Křenek et al. L&B Developer's Guide. http://egee.cesnet.cz/en/JRA1/LB/.

[4] A. Křenek et al. L&B Test Plan. http://egee.cesnet.cz/en/JRA1/LB/.

[5] G. Avellino et al. The DataGrid Workload Management System: Challenges and Results. *Journal of Grid Computing*, 2(4):353–367, Dec 2004.

[6] T. Dierks and C. Allen. The TLS Protocol Version 1.0. IETF RFC 2246 (Standards Track), January 1999.

[7] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastructure (PKI) proxy certificate profile. IETF RFC 3820, June 2004.

[8] A. McNab and S. Kaushal. Web services with gridsite and c/c++/scripts. In *Computing in High Energy and Nuclear Physics (CHEP 2006)*, 2006.

[9] Miroslav Ruda, Aleš Křenek, Miloš Mulač, Jan Pospíšil, and Zdeněk Šustr. A uniform job monitoring service in multiple job universes. In *GMW '07: Proceedings of the 2007 workshop on Grid monitoring*, pages 17–22, New York, NY, USA, 2007. ACM.

[10] Daniel Kouril, Ludek Matyska, and Michal Prochazka. A robust and efficient mechanism to distribute certificate revocation information using the grid monitoring architecture,. In *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, pages 614–619, 2007.

[11] F. Pacini et al. WMS User's Guide. https://edms.cern.ch/file/572489/1/WMS-guide.pdf.