Current and Foreseen L&B Design

CESNET JRA1 team

Introduction

Logging and Bookkeeping service (L&B)

- deployed since first EDG release, stable and reliable
- strongly specialised to track EDG/gLite WMS jobs
- requests to use L&B for "similar tasks"
  - file transfers
  - resource reservations

Talk overview

- current (gLite 1.0) implementation
  - L&B architecture overview
  - API usage
- foreseen changes for gLite 2.x towards "generic L&B"

Introduction

Logging and Bookkeeping service (L&B)

- deployed since first EDG release, stable and reliable
- strongly specialised to track EDG/gLite WMS jobs
- requests to use L&B for "similar tasks"
  - file transfers
  - resource reservations

Talk overview

- current (gLite 1.0) implementation
  - L&B architecture overview
  - API usage
- foreseen changes for gLite 2.x towards "generic L&B"

L&B in gLite 1.0

what is really there

L&B data

Jobs

- primary entity of interest
- assigned unique identifier
- all data in L&B are related to jobs

Events

- record information on points of interest in job life
- the only way to enter information into L&B
- specific types (eg. Transfer, Match, Done)
- carry additional, both generic and type-specific attributes (eg. timestamp, destination CE)
- both "system" (job life) and user (arbitrary info)

L&B data

Jobs

- primary entity of interest
- assigned unique identifier
- all data in L&B are related to jobs

Events

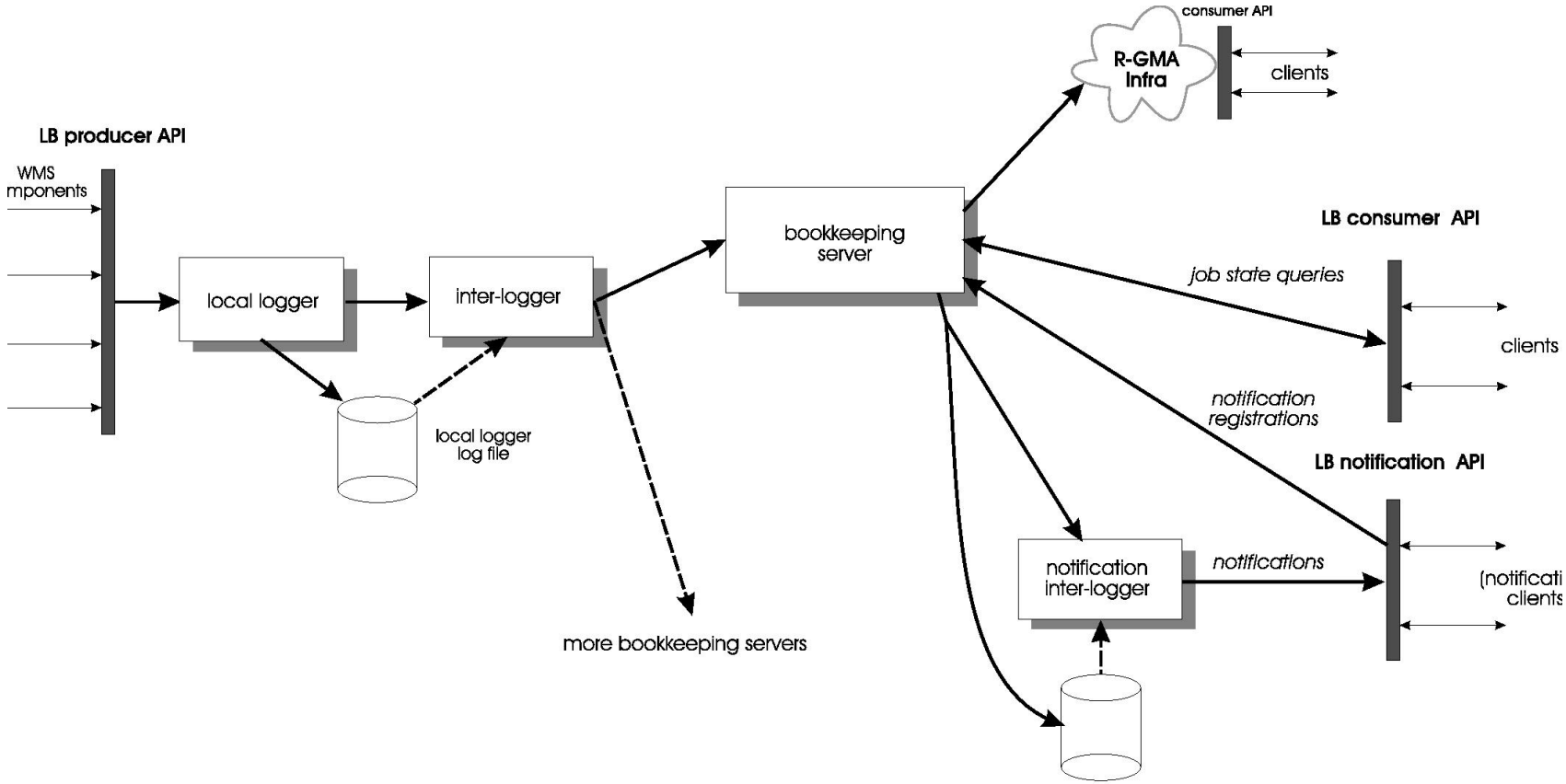
- record information on points of interest in job life
- the only way to enter information into L&B
- specific types (eg. Transfer, Match, Done)
- carry additional, both generic and type-specific attributes (eg. timestamp, destination CE)
- both "system" (job life) and user (arbitrary info)

L&B data (2)

Job states

- eg. Submitted, Running, Done
- computed from events as they arrive
- hard-coded **state machine**
- event attributes mapped to job-state ones
- fault-tolerance – missing or delayed events

L&B components



L&B components (2)

Event sources

- WMS components (and user application)
- instrumented with L&B library calls
- format L&B event, connect to locallogger, authenticate with X509

Local- and interlogger

- add information on authenticated user
- store the event into local file
- confirm accepting the event to the caller
- repeatedly attempt to deliver events to L&B servers
- re-read local files on crash recovery

L&B components (3)

Bookkeeping server

- accept events from interlogger
- store events in raw form
- update job state on event arrival
- answer queries and generate notifications

Consumers

- one-time queries on job state or raw events
  - all my running jobs
  - jobs that failed at CE X last week
- notifications on specified job states changes
  - currently restricted on concrete jobid(s)
  - eg. tell me whenever one of these twenty jobs fails

Event processing

Sequence codes

- correct event order is critical for computing job state
- event delivery is asynchronous, clocks may not be synchronised
- job resubmissions: events ordered in a tree rather than linearly
- events ordered with **hiereachical sequence code**

State machine

- current state + arrived event $\longrightarrow$ new state
- mapping event attributes to job state ones
- fault tolerance – deal with delayed or lost events
- currently hard-coded, C code working on C datatypes

Event processing

Sequence codes

- correct event order is critical for computing job state
- event delivery is asynchronous, clocks may not be synchronised
- job resubmissions: events ordered in a tree rather than linearly
- events ordered with **hiereachical sequence code**

State machine

- current state + arrived event $\longrightarrow$ new state
- mapping event attributes to job state ones
- fault tolerance – deal with delayed or lost events
- currently hard-coded, C code working on C datatypes

Security

- all connections authenticated and encrypted (GSS over SSL)
- clients (data sources and consumers) authenticate with user certs.
- interlogger and lbserver use service (host) certs.
- locallogger adds auth. user info into the events
- users may augment job ACL (add other users or VOMS groups)
- consumers are authorised wrt. job ACL

API usage – producer

- initialise **L&B context** (`SetLoggingJob` API call) with
  - type of the logging component, eg. UI, WM, ...
  - job identifier
  - current sequence code received with the job
- log events reflecting what this component does (`Log<EventType>`)
- extract current seq. code after logging the last event (`GetSeqenceCode`)
- pass on the seq. code with the job

API usage – consumer

- create a set of `QueryRec` structures
  - attribute, eg. jobid, owner, exit code, start-execution time
  - operator: equal, not equal, less, greater, between
  - value
- arange in two-level AND-OR formula (with certain restrictions)
  - (owner = 'me' OR owner = 'my friend')
    AND (start-time between 'Apr 18 9am', 'Apr 22 1pm')
    AND (status = 'Aborted')
- call `QueryJobs` or `QueryEvents` for one-time query
- call `NotifNew` to register for receiving notifications
- see User's guide for more examples and API reference
  - `https://edms.cern.ch/document/571273`

L&B in gLite 2.x

how it may look like

Skeleton and flesh

Principal idea

- keep the current structure – general reusable **skeleton**
- allow applications (legacy L&B, DM, resource reservations) to provide **specific flesh**

What is skeleton

- **job** is the main entity of interest
  - "job" (or "grid process") gets generalised meaning (WMS job, file-transfer job, resource reservation) stored in L&B
  - no anonymous (non-job) information stored in L&B
- jobs are tracked in terms of **events**
- events come from **sources**, reliable **loggers** transfer them to **server**
- server computes **job states**
- users (consumers) pose **queries** or receive **notifications**

Skeleton and flesh

Principal idea

- keep the current structure – general reusable **skeleton**
- allow applications (legacy L&B, DM, resource reservations) to provide **specific flesh**

What is skeleton

- **job** is the main entity of interest
  - "job" (or "grid process") gets generalised meaning (WMS job, file-transfer job, resource reservation) stored in L&B
  - no anonymous (non-job) information stored in L&B
- jobs are tracked in terms of **events**
- events come from **sources**, reliable **loggers** transfer them to **server**
- server computes **job states**
- users (consumers) pose **queries** or receive **notifications**

Skeleton and flesh

Principal idea

- keep the current structure – general reusable **skeleton**
- allow applications (legacy L&B, DM, resource reservations) to provide **specific flesh**

What is flesh

- concrete event and job state datatypes
- plugins for L&B components
  - logging library: functions for logging concrete event types
  - logger: parse and check events, find out destination
  - server: process events to job states
  - query and notification library: attributes for query construction

Most of the plugins code **is generated automatically** from declarative flesh definition.

Skeleton and flesh

Principal idea

- keep the current structure – general reusable **skeleton**
- allow applications (legacy L&B, DM, resource reservations) to provide **specific flesh**

What is flesh

- concrete event and job state datatypes
- plugins for L&B components
  - logging library: functions for logging concrete event types
  - logger: parse and check events, find out destination
  - server: process events to job states
  - query and notification library: attributes for query construction

Most of the plugins code **is generated automatically** from declarative flesh definition.

Flesh definition

- event and job state datatypes defined with XML Schema
  - both event and job state are conforming XML documents
- fixed common "core L&B" schema
  - mandatory events, eg. transfering job between components
  - mandatory event and job state attributes, eg. JobId, timestamp, authenticated user, job location, ...
  - miscelaneous types, eg. sequence code
  - frameworks for job-subjob relationship etc.
- application L&B schema extends the core with its specific types
  - event sources
  - event types
  - job states
  - job state attributes
  - ...

Flesh definition

- event and job state datatypes defined with XML Schema
  - both event and job state are conforming XML documents
- fixed common "core L&B" schema
  - mandatory events, eg. transfering job between components
  - mandatory event and job state attributes, eg. JobId, timestamp, authenticated user, job location, ...
  - miscelaneous types, eg. sequence code
  - frameworks for job-subjob relationship etc.
- application L&B schema extends the core with its specific types
  - event sources
  - event types
  - job states
  - job state attributes
  - ...

Flesh definition

- event and job state datatypes defined with XML Schema
  - both event and job state are conforming XML documents
- fixed common "core L&B" schema
  - mandatory events, eg. transfering job between components
  - mandatory event and job state attributes, eg. JobId, timestamp, authenticated user, job location, ...
  - miscelaneous types, eg. sequence code
  - frameworks for job-subjob relationship etc.
- application L&B schema extends the core with its specific types
  - event sources
  - event types
  - job states
  - job state attributes
  - ...

L&B logger

- current local- and interlogger merged (maybe) into a single component
- WS interface for logging
  - identical for asynchronous (via logger) and synchronous (directly to server) logging
  - client indepence on logging library implementation
  - optional destination L&B server address
  - additional parameters: synchronicity, priority, ...
- besides core L&B fields logger does not understand the event contents
- events stored in logger's files "as is", ie. XML
- optional application plugin
  - parse and check events
  - extract destination L&B server address

L&B logger

- current local- and interlogger merged (maybe) into a single component
- WS interface for logging
  - identical for asynchronous (via logger) and synchronous (directly to server) logging
  - client indepence on logging library implementation
  - optional destination L&B server address
  - additional parameters: synchronicity, priority, ...
- besides core L&B fields logger does not understand the event contents
- events stored in logger's files "as is", ie. XML
- optional application plugin
  - parse and check events
  - extract destination L&B server address

L&B logger

- current local- and interlogger merged (maybe) into a single component
- WS interface for logging
  - identical for asynchronous (via logger) and synchronous (directly to server) logging
  - client indepence on logging library implementation
  - optional destination L&B server address
  - additional parameters: synchronicity, priority, ...
- besides core L&B fields logger does not understand the event contents
- events stored in logger's files "as is", ie. XML
- optional application plugin
  - parse and check events
  - extract destination L&B server address

Logging library

- core logging library and application plugin
- not mandatory – event sources may implement independent logger WS client
- convenience functions similar to current library:
  - L&B context handling (core)
  - sequence codes (core)
  - logging of concrete event types (plugin, generated)
- binding in C, later C++ and Java (if required)

L&B server – core functionality

- multiprocess server, master-to-slave request dispatching
- WS interface
  - incoming (logging) – identical with logger
  - queries and notifications
- database management
  - events and job states storage (XML)
  - specific attributes extracted to separate columns to allow queries and indexing
- query processing
  - validation vs. indices
  - translation to SQL
  - imposing limits on result-set size
- authorisation
- generating notifications

L&B server – application plugin

- event and job state serialisation / deserialisation–parsing (generated)
- event ordering function (seqence codes, timestamps, . . .)
- terminal state detection and purging strategy
- queries and notifications: datatypes, attributes
- extended querying capabilities (post-EGEE)
- job state machine (details follow)

L&B server – state machine

- application specific (ie. inside plugin)
- mapping: current state + new event $\longrightarrow$ new state (including attribute translation)
- may ask server core for additional events
- final goal (post-EGEE)
  - describe the machine in declarative way
  - most of code generated automatically (fault-tolerance, attribute transformation, . . .)
  - call hand-written "refinement hooks"
  - we need more usecases, not only current L&B
- intermediate implementation (EGEE)
  - all functionality in the hooks

Issues

Application-specific data structures

- let gSoap et al. generate them from .xsd completely
  - likely to introduce dependence on concrete gSoap version
- independent XML parser (using libxml etc.)
  - more work but more portable
- to be stable, all plugin interfaces must be neutral
  - arguments are XML strings only
  - parsing and serialisation done inside plugin

Out-of-order events

- a delayed event arrives
- state machine should ignore it wrt. state computation
- the event still may carry useful attributes
- must be handled in the state machine

Issues

Application-specific data structures

- let gSoap et al. generate them from .xsd completely
  - likely to introduce dependence on concrete gSoap version
- independent XML parser (using libxml etc.)
  - more work but more portable
- to be stable, all plugin interfaces must be neutral
  - arguments are XML strings only
  - parsing and serialisation done inside plugin

Out-of-order events

- a delayed event arrives
- state machine should ignore it wrt. state computation
- the event still may carry useful attributes
- must be handled in the state machine

Issues (2)

Dynamic event ordering

- "shallow resubmit" WMS usecase
- different branches of events may become active
- how to reflect it in state-machine interface?

Multiple plugins in one server

- eg. both WMS jobs and resource reservations
  - should be supported in general
- shall we allow combined queries like "what are my waiting jobs with expired resource reservations"?
  - preferably not, task of RGMA :-)

Many more, yet unknown . . .

Issues (2)

Dynamic event ordering

- "shallow resubmit" WMS usecase
- different branches of events may become active
- how to reflect it in state-machine interface?

Multiple plugins in one server

- eg. both WMS jobs and resource reservations
  - should be supported in general
- shall we allow combined queries like "what are my waiting jobs with expired resource reservations"?
  - preferably not, task of RGMA :-)

Many more, yet unknown . . .

Issues (2)

Dynamic event ordering

- "shallow resubmit" WMS usecase
- different branches of events may become active
- how to reflect it in state-machine interface?

Multiple plugins in one server

- eg. both WMS jobs and resource reservations
  - should be supported in general
- shall we allow combined queries like "what are my waiting jobs with expired resource reservations"?
  - preferably not, task of RGMA :-)

Many more, yet unknown ...

Preliminary conclusions

- The requirements cannot be satisfied with “quick hacks”.
- Non-trivial design changes required.
- Many challenges, non-trivial amount of work
  - but not too much code written from scratch
  - cooperation from non-WMS applications required
- We are willing to take this way.
- Commitment of potential “customers” is necessary.

Preliminary conclusions

- The requirements cannot be satisfied with “quick hacks”.
- Non-trivial design changes required.
- Many challenges, non-trivial amount of work
  - but not too much code written from scratch
  - cooperation from non-WMS applications required
- We are willing to take this way.
- Commitment of potential “customers” is necessary.

Preliminary conclusions

- The requirements cannot be satisfied with “quick hacks”.
- Non-trivial design changes required.
- Many challenges, non-trivial amount of work
  - but not too much code written from scratch
  - cooperation from non-WMS applications required
- We are willing to take this way.
- Commitment of potential “customers” is necessary.