Job-centric Monitoring in gLite

Stefan Borovac, Torsten Harenberg,
Peter Mättig, **Markus Mechtel**,
David Meder-Marouelli

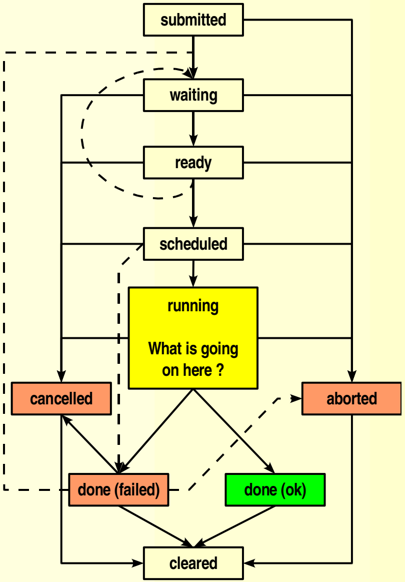
University of Wuppertal

June 20th, 2007

Outline

Motivation

- gLite jobs end with Status done(success) or done(failed)
- ~20% of all jobs fail
- gLite middleware reports status of grid-infrastructure (transport layer), not result of jobs
- user does not get any information about sources of job failures



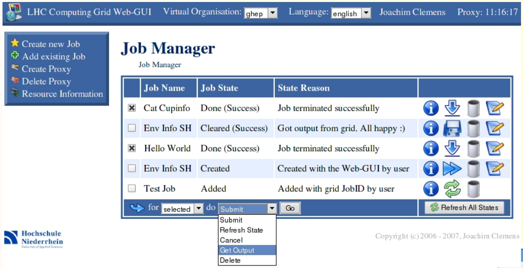
Sources of Failures

- Grid middleware misconfiguration
- external services not available
- Worker node configuration
- Worker node problems
  - full hard disk
  - network connection missing
  - hardware defects
  - Firewall misconfiguration
- missing software (e.g. needed libraries)
- errors in user software
- . . .

Solution: Job Execution Monitor

- Job monitoring on worker node
  - stepwise execution of scripts
  - monitoring of executed commands
- Realtime Information
  - User knows current state of his jobs
  - access to stdout/stderr output even in case of errors (not available in gLite with failed jobs)
- graphical Interface for clear Display in GridSphere

Screenshots

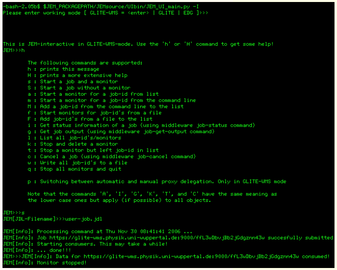


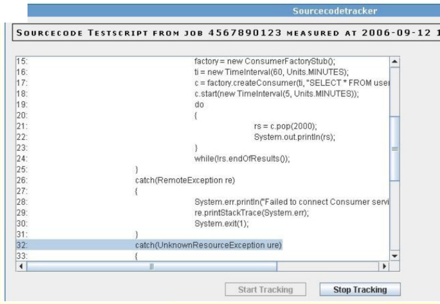
Job Manager

Job Manager

	Job Name	Job State	State Reason
☒	Cat Cupinfo	Done (Success)	Job terminated successfully
☐	Env Info SH	Cleared (Success)	Got output from grid. All happy :)
☒	Hello World	Done (Success)	Job terminated successfully
☐	Env Info SH	Created	Created with the Web-GUI by user
☐	Test Job	Added	Added with grid JobID by user

Copyright (c) 2006 - 2007, Joachim Clemens


```
-bash-2.05b$ $JEM_PACKAGEPATH/JEMsource/UIbin/JEM_UI_main.py -I
Please enter working mode [ GLITE-WMS = <enter> | GLITE | EDG ]>>>



This is JEM-interactive in GLITE-WMS-mode. Use the 'h' or 'H' command to get some help!
JEM>>>h

        The following commands are supported:
        h : prints this message
        H : prints a more extensive help
        s : Start a job and a monitor
        S : Start a job without a monitor
        a : Start a monitor for a job-id from list
        m : Start a monitor for a job-id from the command line
        M : Add a job-id from the command line to the list
        f : Start monitors for job-id's from a file
        F : Add job-id's from a file to the list
        i : Get status information of a job (using middleware job-status command)
        g : Get job output (using middleware job-get-output command)
        l : List all job-id's/monitors
        k : Stop and delete a monitor
        t : Stop a monitor but left job-id in list
        c : Cancel a job (using middleware job-cancel command)
        w : Write all job-id's to a file
        q : Stop all monitors and quit

        p : Switching between automatic and manual proxy delegation. Only in GLITE-WMS mode

        Note that the commands 'A', 'I', 'G', 'K', 'T', and 'C' have the same meaning as
        the lower case ones but apply (if possible) to all objects.

JEM>>>s
JEM[JDL-Filename]>>>user-job.jdl

JEM[Info]: Processing command at Thu Nov 30 08:41:41 2006 ...
JEM[Info]: Job https://glite-wms.physik.uni-wuppertal.de:9000/ffL3wDbvjBb2jGdgznn43w succesfully submitted
JEM[Info]: Starting consumers. This may take a while!
JEM[Info]: ... done!!!
JEM>>>JEM[Info]: Data for https://glite-wms.physik.uni-wuppertal.de:9000/ffL3wDbvjBb2jGdgznn43w consumed!
JEM[Info]: Monitor stopped!
```



SOURCECODE TESTSCRIPT FROM JOB 4567890123 MEASURED AT 2006-09-12 1

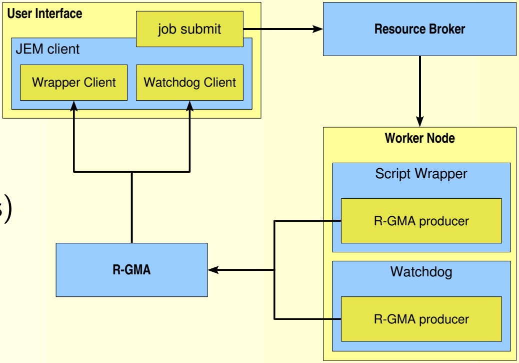
```
                    factory = new ConsumerFactoryStub();
                    ti = new TimeInterval(60, Units.MINUTES);
                    c = factory.createConsumer(ti, "SELECT * FROM use
                    c.start(new TimeInterval(5, Units.MINUTES));
                    do
                    {
                              rs = c.pop(2000);
                              System.out.println(rs);
                    }
                    while(!rs.endOfResults());
          }
          catch(RemoteException re)
          {
                    System.err.println("Failed to connect Consumer servi
                    re.printStackTrace(System.err);
                    System.exit(1);
          }
          catch(UnknownResourceException ure)
          {
```

Architecture

- Python
  - installed on every gLite node
  - plattform independant (automatically runs on 64bit CPUs)
- Information exchange exclusively via the Relational Grid Monitoring Architecture (R-GMA)
  - no firewall problems

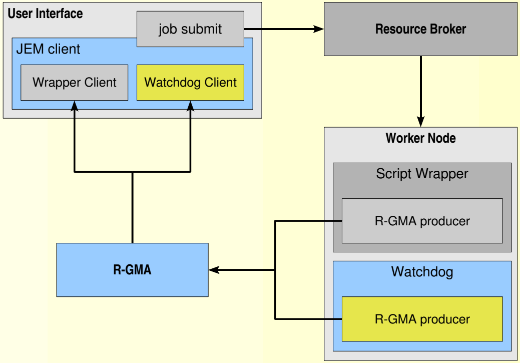
Strukture

- command line interface/menu
- JEM automatically added to job
  → no additional work
- 2 components
  - Watchdog
  - Script Wrapper
- stepwise execution of
  Bash- and Python-Scripts
- regular status messages
  via R-GMA
  (job status, system resources)
- graphical Display of
  system resources on the UI
- detailed logfile

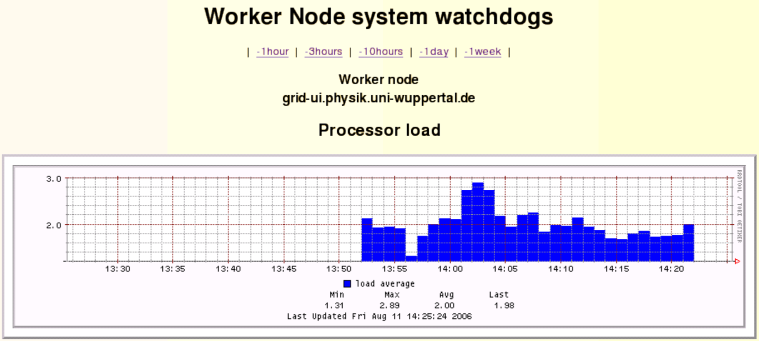


Watchdog

- monitors system resources
  - free RAM
  - free disk space
  - network traffic
  - processor load
- Daten published regularly via R-GMA
- graphical Display



Watchdog



Worker Node system watchdogs

| -1hour | -3hours | -10hours | -1day | -1week |

Worker node
grid-ui.physik.uni-wuppertal.de

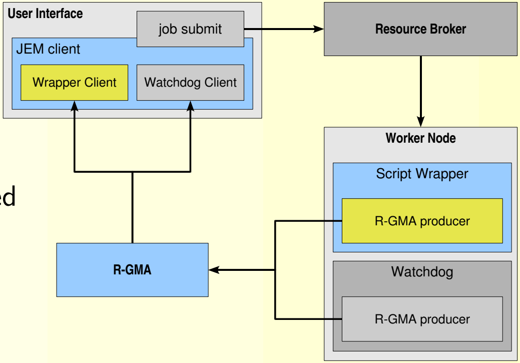
Processor load

```
■ load average
  Min     Max     Avg     Last
  1.31    2.89    2.00    1.98
Last Updated Fri Aug 11 14:25:24 2006
```

- trend display of system resources

Script Wrapper

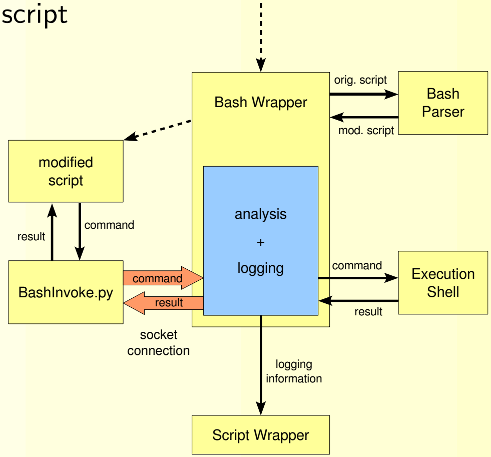
- stepwise execution
  → backtrace of commands in case of errors
- known-critical commands may be modified/hardened
- Data published regularly via R-GMA
- supported languages:
  - Bash
  - Python
- modules for additional languages can be easily added



Bash Wrapper

operation principle

- Parser identifies commands in script
- Wrapper starts isolated shell
- Wrapper starts modified script
- modified script sends single command to subshell
- Execution shell runs command
- Wrapper monitors and logs results



Python Wrapper

- Python provides mechanisms for monitoring Python commands
- operation principle
  - read environment variables from execution shell
  - monitor execution of Python script
  - write back (modified) environment variables to execution shell

Job Execution Monitor

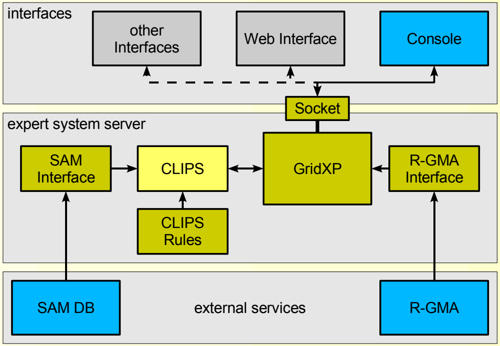
- monitors system resources
- monitors execution of script files
- reports status via R-GMA
- much additional information about job execution
- information used by expert system

Expert system

helps finding and fixing job failures and error conditons

Architecture:

- client-server architecture
- CLIPS expert system shell as backend
- client-server connection via socket
- many client interfaces possible
  - command line
  - web interface
  - ...

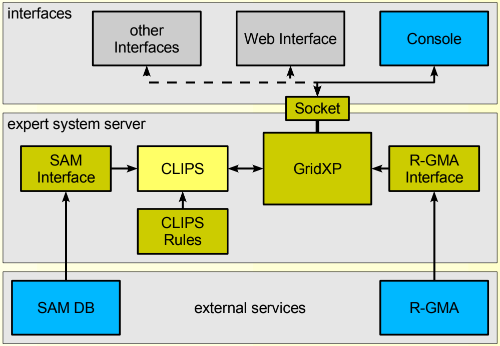


Expert system

sources of data

- R-GMA
  - data retrieval takes time
  - data continuously fed in
  - often not accessible
- Service Availability Monitor (SAM)
  - central database at CERN
  - data queried when needed
  - access restricted to known IP addresses

CLIPS rules combine information from different sources



Summary

reached goals

- stepwise execution of Bash- und Python-scripts
- Wrapper for additional Languages may be easily added
- Backtracing of errors is possible
- monitoring of system resources on worker node
- a lot of additional information about job execution
- integration into GANGA with athena jobs
- prototype of expert system

JEM download at

http://www.grid.uni-wuppertal.de/jem

Outlook

Job Execution Monitor

- complete Bash syntax
- integrate into existing monitoring tools (Ganga, gLite, . . .)

Expert system

- definition of rules
- classification of job failures
- looking for additional sources of data

Thanks for your attention