Monitoring system basics, and adding support for a new batch system

September 19, 2012  |  Carsten Karbach and Wolfgang Frings
Content

1. PTP System Monitoring
2. Architecture
3. Large-scale system Markup Language (LML)
4. Implementation
5. Scalability
6. Add new target system
7. Conclusion
Part I: PTP System Monitoring

September 19, 2012 | Carsten Karbach and Wolfgang Frings
PTP Monitoring Scope

- job and system monitoring of large-scale supercomputers
- monitoring of multiple target systems in one perspective
- support for many batch systems (Grid Engine, LoadLeveler, Open MPI, PBS, Slurm, Torque)
- overview of the system on a single screen
- based on monitoring application LLview
1. PTP System Monitoring

PTP Monitoring Perspective
PTP Monitoring Perspective

Job Views

1. PTP System Monitoring
1. PTP System Monitoring

PTP Monitoring Perspective

Monitoring View
PTP Monitoring Perspective
Monitoring Views

- **Nodes View** renders target system architecture, maps jobs to compute resources
- **Active Jobs View** lists running jobs
- **Inactive Jobs View** lists queued jobs
- **Monitoring View** selects active target system, starts/stops monitoring
- **Message View** shows message of the day
Part II: Architecture

September 19, 2012  |  Carsten Karbach and Wolfgang Frings
Monitoring Architecture I
Monitoring Architecture II

- **LML-da** gathers status information, calls target system’s remote commands, written in Perl
- **LML** is a data format for status information of supercomputers
- LML request: contains table filtering information, visible/hidden columns
- LML response: contains the request and status information
- client stores **current layout** request for successive Eclipse sessions
Communications Protocol

: LML\_da \rightarrow \text{send LML\_da to server} \rightarrow \text{Client}

\text{empty request} \rightarrow \text{LML file} \rightarrow \text{store}

\text{LML-Layout} \rightarrow \text{modify} \rightarrow \text{load} \rightarrow \text{replace}

\text{request update} \rightarrow \text{LML file} \rightarrow \text{load} \rightarrow \text{replace}

\text{loop}
Part III: Large-scale system Markup Language (LML)

September 19, 2012  |  Carsten Karbach and Wolfgang Frings
Large-scale system Markup Language (LML)

- markup language for description of supercomputer’s status
- **interface** between LML-da and visualization clients
- describes all **graphical components** available in LLview (job-list, node display, charts ...)
- logical and system independent description of current status
- close to graphical representation → thin visualization clients
- implemented in XML, validation against XML-Schema
LML Structure

- **Global data**: intermediate data format, scheduling objects
- **Graphical components**: data for visualization components
- **Layout**: hints for visualization, node display hierarchy
Example node display

LML-description of a node display

```xml
<nodedisplay title="Cluster" id="1">

<scheme>
    <el1 tagname="Node" min="1" max="4">
        <el2 tagname="CPU" min="1" max="3"/>
    </el1>
</scheme>

<data>
    <el1 min="1" max="2" oid="job1">
        <el2 min="3" oid="job2"/>
    </el1>
    <el1 min="3" max="4" oid="job2"/>
</data>

</nodedisplay>
```
Part IV: Implementation

September 19, 2012 | Carsten Karbach and Wolfgang Frings
4. Implementation

Plug-in Overview

- **lml.core**
  - handles LML
  - UI independent
  - extracts of LML layout

- **lml.ui**
  - visualizes LML components
  - collects infos from lml.core

- **monitor.core**
  - provides actions for each connection (start, stop, refresh)

- **monitor.ui**
  - implements Monitoring View
  - calls monitor.core for user actions

- **lml.da**
  - Perl scripts executed on remote system

- **lml.da.server**
  - calls lml.da for LML requests
  - generates lml-driver.tar
## Plug-in Details I

### lml.core
- uses **JAXB** to manage LML files
- provides **helper classes** for comfortable access on LML
- defines **events and listeners** for all UI actions

### lml.ui
- implements a **ViewPart** for Nodes / Jobs and Info Views
- renders LML components, considers LML layout hints
- UI implementation:
  - Jobs View → JFace TreeViewer
  - Nodes View → nests Composites, PaintListener
  - Info View → SWT Text / Table
## Plug-in Details II

**lml.da**

- set of Perl scripts → simple installation
- split into independent modules:
  1. **Driver** calls system specific commands to get status infos
  2. **Combiner** merges LML files generated by the Driver
  3. **AddColor** assigns unique color to each job
  4. **LML2LML** converts intermediate format to final LML

- fully configurable workflow
- only Driver scripts and workflow description are target system specific
Part V: Scalability
5. Scalability

Node Display – Compression

System

Rack1
- Nodecard1
- Nodecard2
- Nodecard3

Rack2
- Nodecard1
- Nodecard2
- Nodecard3

Rack3
- Nodecard1
- Nodecard2
- Nodecard3

Rack4
- Nodecard1
- Nodecard2
- Nodecard3
Node Display – Compression
Attribute Inheritance
Node Display – Compression

Scheme defines architecture of empty system
Node Display – Compression

Ranges

Result

→ 3 objects instead of 16
Scalable Visualization – Row Level
Scalable Visualization – Nodeboard Level

- JUQUEEN
  - Row 1
  - Row 2
  - Rack 1..4
  - Midplane 1..2
  - Nodeboard 1..16
5. Scalability

Scalable Visualization – CPU Node Level

- JUQUEEN
- Row 1
- Row 2
- Rack 1..4
- Midplane 1..2
- Nodeboard 1..16
- CPU Node 1..32
Table Filtering

- scalability of job data display
- filtering by specification of attribute values / ranges
- server- and client-side filtering
Part VI: Add new target system
LML\textsubscript{da} – Data Extraction

Which status data is required?

- **jobs**: owner, queue, dispatch date, state ...
- **nodes**: memory, cores, state, GPUs ...
- **system**: hostname, date, high messages ...

How is status data extracted?

- one Perl script per data type
- call batch system commands (e.g. `qstat` and `pbsnodes` for TORQUE)
- convert output into intermediate LML files
LML_da – add new target system

1. write scripts for extracting status information e.g.
   - `da_jobs_info_LML.pl` gathers jobs
   - `da_nodes_info_LML.pl` gathers nodes
   - `da_system_info_LML.pl` gets global system information

2. write functions for checking remote commands and for composing the workflow → `da_check_info_LML.pl`

3. put all scripts into a folder in `lml.da/rms`, folder named after target system

4. add `lml.monitor.ui.monitors` extension for the new system type
Client – add new target system

- LML_da generates **system independent** LML files
- client visualizes new target systems without any changes
- **but** a system specific LML layout can be configured
- currently this layout configuration is tricky
LML Layout Configuration I

- the following procedure is only meant for **testing**
- **client side configuration** of LML layout is planned
- **long-term target:**
  configure entire layout via GUI, not via XML file
- the layout could be saved together with the target system configuration
6. Add new target system

LML Layout Configuration II

1. write your own LML layout file, or adjust an existing (examples in lml.da plug-in)
2. start the monitoring connection to the target system
3. log-in on the remote machine and switch to the .eclipsesettings folder in your home directory
4. create the file .LML_da_options with content nocheckrequest=1
5. replace samples/layout_default.xml in .eclipsesettings with your own layout
6. refresh the monitoring perspective on your client
6. Add new target system

Layout Example – Default
6. Add new target system

Layout Example – Custom I
6. Add new target system

Layout Example – Custom II
Part VII: Conclusion

September 19, 2012  |  Carsten Karbach and Wolfgang Frings
Conclusion

- 5 interacting views form the **monitoring perspective**
- **LML** as data interface between LML_da and client
- LML is divided into global data, graphical components and layout
- composition of 6 Eclipse plug-ins (lml.core/ui, monitor.core/ui, lml.da/lml.da.server)
- **new target system**: write data extraction scripts + configure layout
- Scalability
  - scalable data **acquisition**
  - scalable data **format**
  - scalable **presentation**
Questions?
Contact

- **E-mail:**
c.karbach@fz-juelich.de, w.frings@fz-juelich.de

- **LML** → http://llview.zam.kfa-juelich.de/LML

- **LLview** → http://www.fz-juelich.de/jsc/llview