Performance testing of distributed computational resources in the software development phase

J. Cernak, E. Cernakova and M. Kocan,

P. J. Safarik University in Kosice, Slovak Republic

EMI is partially funded by the European Commission under Grant Agreement RI-261611

Outline

- Performance testing-basic terms, motivations and requirements
- Tools for performance testing
- Test cases in EMI for ARC components
- Conclusions

Performance testing-basic

- **Performance testing** is testing to determine a response and stability of system under specific workload
- It can cover investigation, measurement or verification of:
  - quality attributes:
    - *scalability,*
    - *Reliability (started before EMI 1.0.0) ,*
    - *duration of events (started before EMI. 1.0.0),*
  - resource usage (started before EMI 2.0.0).

Performance testing-motivation

- Factors which can influence performance
  - implementation of common standards in existing middleware solutions can increase complexity of the software,
  - core middleware software depends on many software solutions which could affect the final parameters.
- Reasons to introduce performance testing:
  - to provide detail feedback for developers about possible performance issues in early state of development,
  - an effort to produce high quality software-be compliant with EMI policies [6].

Performance testing-requirements

- **Load testing**-investigate performance parameters in specific load, for example extensive job submission.
  - Stress testing-investigate upper limits for load testing.
- **Endurance testing**-monitoring of memory or CPU usage to detect memory leaks or CPU workload during load testing.
- **Parameter testing**-searching for the optimal setting to ensure the best performance results.
- **Setting the performance goals**:
  - to define performance criteria,
  - compare software parameters between release candidates or other software solutions,
  - determine a weak part of software.

Performance testing tools

- We have identified two major testing tools:
  - **mature tools** deployed to monitor daily operation of:
    - *Networks (MRTG),*
    - *Servers(Ganglia, Nagios, Monalisa),*
    - *web services(Nagios, Monalisa);*
  - **custom solutions** - set of specific scripts (.sh, .py) and visualisation tools
- Both solutions could be used in the software development phase, however in the development phase specific needs exist to provide more detail information about software properties, for example activity of process.

Performance tools-cont.

- Initially we used custom bash scripts (.sh). Required information about process activity was provided by Linux **ps** utility. A lot of manual work was needed to visualise the results [1,2].
- At present we proposed a solution based on utilisation of cross platform **psutil** module for Python [3], **MySQL** database and control layer (**xlm-rpc**). We proposed a web page [4] with PHP interface to DB and jqplot to plot graphs of the test results.
- The performance tool connects an advantage of python module **psutil,** central DB, separated communication layer and power of **jqplot**.

Performance tools-cont.

- Key parts of proposed advanced performance tool:
  - Python script p1.py is used to monitor process activity
  - Python script p2.py is used to send data from monitored distributed resources (server or client) to the central database
  - Control layer is used to synchronise events, for example remote start or stop of scripts p1.py or p2.py.

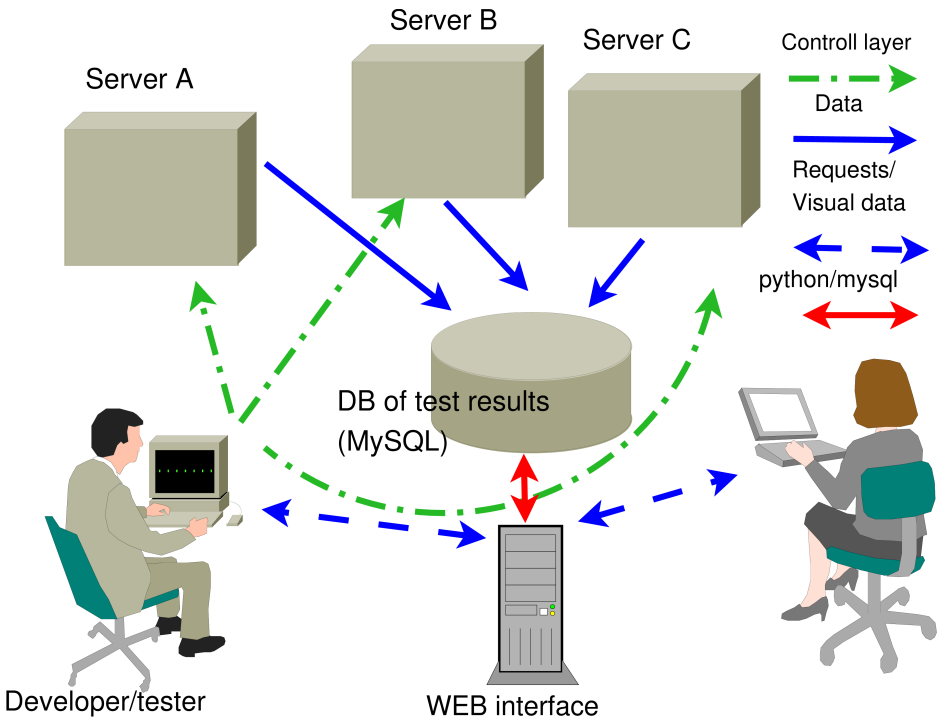
Performance tool-cont.

- p1.py - monitors set of daemons (processes) and their subprocesses
  - The script was tested with:
    - *Python v 2.4.4, v2.6.5,*
    - *psutil v. 0.4.1,*
    - *OS SL5,CentOS5, Ubuntu 10.4.*
  - Script usage -list of process names separated by spaces:
    - *command: nice -n -16 python p1.py "p1 p2 … pn" &*
- p2.py reads temporary file and sends data to central DB.
  - Script usage- list of process names separated by spaces
    - *Command: python p2.py "p1 p2 … pn"*

Performance tool-cont.

- Control layer
  - unsecure communication “client - server” using python xml-rpc module (tested)
  - Plan to use secure communication
- Visualisation of results
  - For each process and sub-processes we can determine six parameters:
    - *mem %, CPU%, CPU usage by user CPU0 and system CPU1, and memories parameters rrs and vms.*
  - Format of the results:
    - ***Iterative graphs (zooming)**, .gif figure and .csv file*

Performance tool-design



Test cases in EMI for ARC components

- **Client-server** performance characteristics of task management (load testing):
  - time to
    - *submit 1000 sequential tasks,*
    - *time to download the results of submitted tasks (1000 tasks)*
  - Reliability (ratio of successful to all tasks) of :
    - *job submission, acceptance criteria R=1,*
    - *job downloading, R=1,*
    - *successful jobs, R=1.*

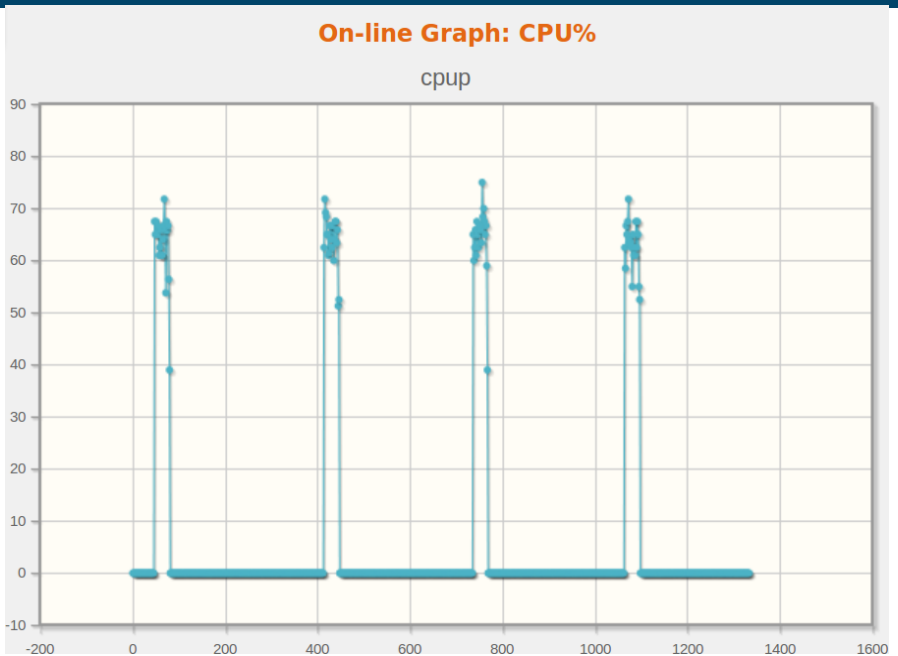
Test cases in EMI for ARC-cont

- Identified issues by using performance testing:
  - Memory leak of CE
  - Reliability testing (increase reliability of job submission EMI 1.0.0)
  - Duration of job submission (reduction of time to submit 1000 jobs)
  - Duration of download 1000 jobs (identified issue)
  - BDII configuration issue (a big memory usage)

Endurance testing

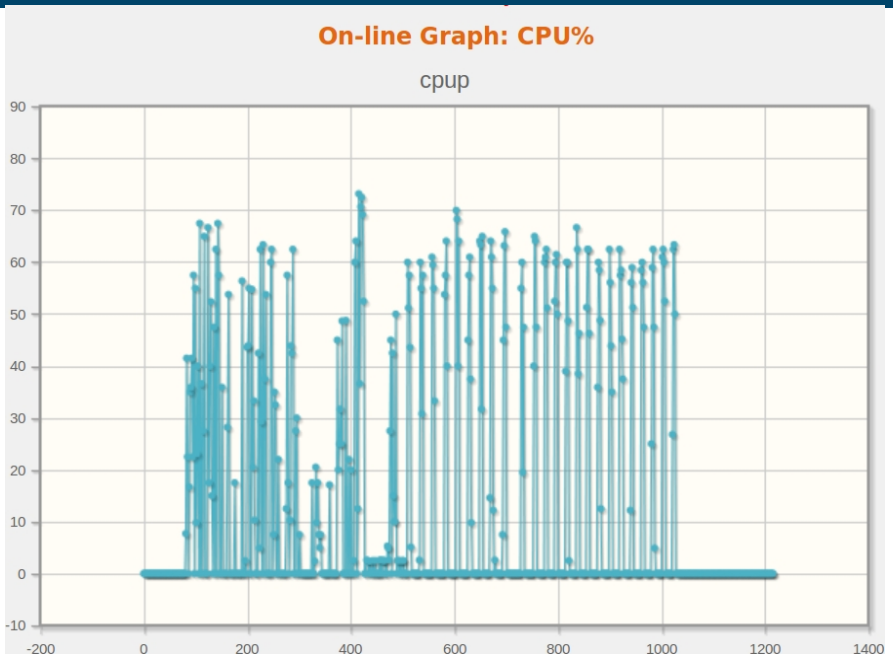
- Obviously two important parameters are monitored:
  - Memory usage
  - CPU usage
- Example of monitoring CPU usage of **slapd** of ARC0 CE:
  - Load characteristics:
    - *no job submission from client*
    - *job submission of 1000 tasks*

Example of endurance testing CPU usage

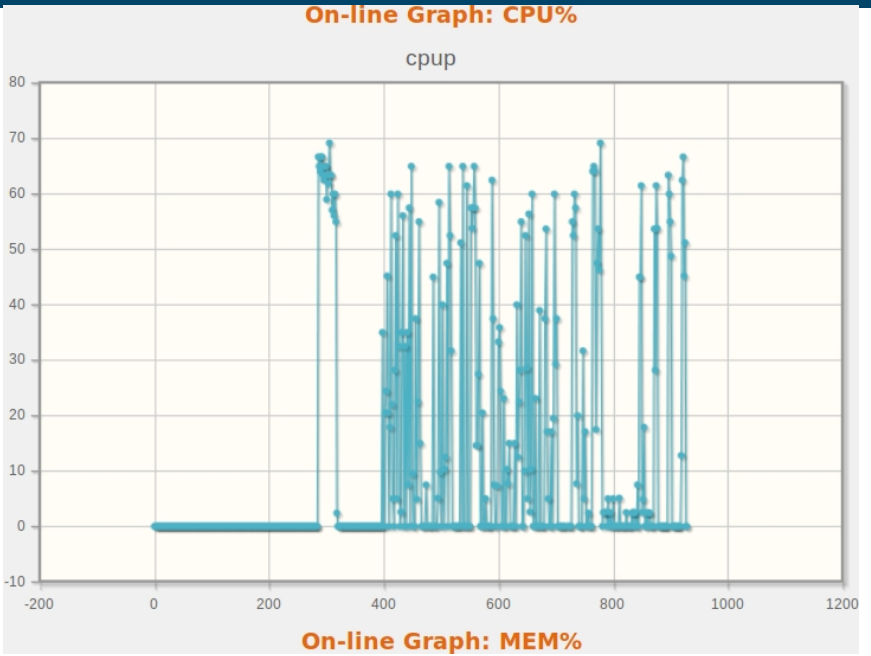


No job submission, however slapd is active

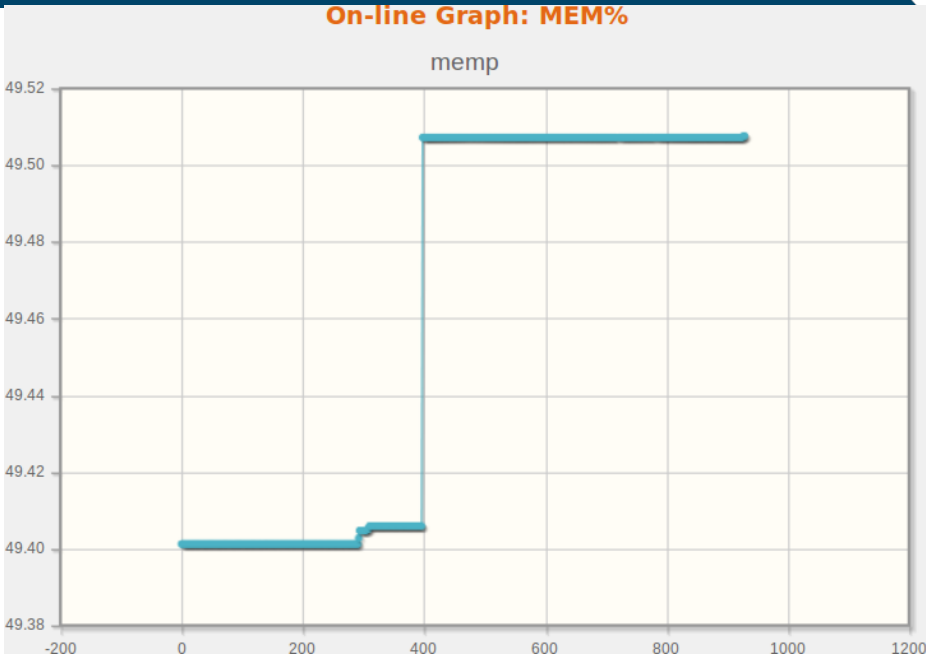
Endurance testing-cont.



slapd activity during 1000 job submission



slapd CPU usage during 1000 job performace test



slpad memory usage during 1000 job submission performance test

Plans:

- to provide the results of performance tests for EMI 2.0.0.
- To extend the number of test scenarios
- Systematic performance analysis after EMI 2.0.0.
- trying to define minimal performance parameters for ARC components.

Conclusions

- Prototype of performance tool:
  - was developed and tested as part of EMI SA2.4 task,
  - the python scripts could be integrated into similar test frameworks,
  - can be used in similar software projects.
- Realisation of performance testing in development phase:
  - can discover potential issues very early,
  - increase a quality of the final software.
- Need of definition of common performance parameters to evaluate a quality of grid software

References

- [1] D5.4-1 RESOURCE CONSUMPTION PROFILE AND PERFORMANCE BENCHMARK STUDY OF THE EARLY PROTOTYPE RELEASE OF KNOWARC: http://www.knowarc.eu/documents/Knowarc_D5.4-1_07.pdf
- [2] D5.4-1 RESOURCE CONSUMPTION PROFILE AND PERFORMANCE BENCHMARK STUDY OF THE FINAL RELEASE OF KNOWARC: http://www.knowarc.eu/documents/Knowarc_D5.4-1_09.pdf
- [3] “psutil” cross-platform process and system monitoring module for Python: http://code.google.com/p/psutil/
- [4] ARC tools for revision, functional and performance testing (including DB of test results and tool to generate EMI test reports): http://arc-emi.grid.upjs.sk/tests.php
- [5] ARC test coordination: http://wiki.nordugrid.org/index.php/Testing
- [6] EMI testing policies: https://twiki.cern.ch/twiki/bin/view/EMI/EmiSa2TestPolicy