A Performance Analysis Tool
for Interactive Applications on the Grid

Roland Wismüller
Institute for Software Science, University of Vienna
LRR-TUM, Technische Universität München

Marian Bubak, Wlodzimierz Funika, Bartosz Balis
ICS-AGH, University of Metallurgy and Mining, Krakow, Poland
Background

CrossGrid Project

- Funded by EU: 03/2002 - 02/2005
- Goal: extend Grid towards *interactive* applications
- [http://www.eu-crossgrid.org](http://www.eu-crossgrid.org)

CrossGrid Applications

- Simulation of vascular blood flow
- Flooding crisis support
- Data analysis in High Energy Physics
- Weather forecast and air pollution
Simulation of vascular blood flow

Interactive visualization and simulation
- response times are critical
- 0.1 sec (head movement) to 5 min (change in simulation)

Performance analysis
- response time and its breakdown
- performance data for specific interactions
Flooding crisis support

Interactive steering
- start / kill simulations (based on new data / performance)

Performance analysis
- convergence behavior, progress of computation
- infrastructure related performance data

User

Meteorological simulation
- Control
- Precipitation

Hydrological simulation
- Discharge

Hydraulical simulation
- Water stage

Parameter studies

Roland Wismüller, Universität Wien / Technische Universität München
On-line Performance Analysis

Properties

- User defines performance *measurements* at run-time
- Results are numerical time series
- Selective run-time instrumentation
- Counters and (integrating) timers, *very* limited tracing

Motivation

- Correlate performance data with interactions
- On-line data necessary for steering
- Reduce the amount of measurement data
- Allow to start monitoring *after* application started
Special Requirements

- Integration with Grid infrastructure
  - e.g. job submission / scheduling

- Acquire infrastructure related performance data
  - steering / resource selection
  - assessment of application’s performance data

- Focus on interaction of distributed application components
  - assumption: components are already optimized

- Provide data meaningful in the context of an application
  - take into account application specific information
Application specific information

Examples

- How much disk I/O was caused by a specific user interaction?
- How much of the response time is due to
  - communication between client and server,
  - computation within client / server,
  - communication inside the server,
  - ...?
- What is the current convergence rate of my solver? (When will the result be available?)
Approach

- Performance analysis tool G-PM offers “standard” metrics
  - e.g. CPU time, communication time, disk I/O; node load
  - automatic instrumentation

- Application programmer provides a minimum amount of information
  - relevant events inside application
  - association between events
  - relevant data computed by the application
  - manual source code instrumentation (probes)

- G-PM allows to define new metrics
  - based on existing ones and application specific information
Example

Disk I/O caused by an interaction (naive)

- Define measurement “disk I/O”
- Probes mark beginning and end of an interaction
- Result = disk I/O (end) - disk I/O (begin)
Problem: Semantics

- In a distributed system, simultaneity is not well-defined.
- So semantics can’t be this:

This is also impossible to implement.
Problem: Semantics

- We should only consider I/O caused by the interaction
- causal relationships (*happened before, →*)
- I/O events causally “between” *begin* and *end*

- **Drawbacks:**
  - on-line event processing no longer possible
  - not correct, if processes use request queues
Problem: Semantics

- Consequence: *begin* and *end* probes locally in each process
- Correspondence indicated by matching IDs passed to probes *(virtual time)*

![Diagram showing begin and end probes locally in each process](image)

- Drawback:
  - more work for the user, if code is non-SPMD
Problem: Loss of Information

- Interesting effects may be hidden by summaries
- E.g. load imbalance (waiting time at barrier in a loop):

![Diagram showing loops and barriers for processes P1 and P2]

- Solution: read a “trace” of timer values: one value for each execution of the loop (*virtual time*)
Design: Components

- Grid Information Services
  - OCM-G
    - Service Manager
      - P1
      - P2
      - P3
      - ... Pn
  - OCM-G Interface (OMIS)
  - Measurement Interface
- User Interface & Visualization Component
- High-Level Analysis Component
- Performance Measurement Component
- G-PM

Roland Wismüller, Universität Wien / Technische Universität München
Design: User Interface

Choose metric
- Send volume
- Receive volume

Selected objects
- Cyfronet
- TUM

Choose hosts
- n_1000
- n_2000

Choose processes
- p_876_n_1000
- p_877_n_1000

Choose sites
- communication

Choose modules
- doStepAlpha
- doStepBeta

Choose hosts
- n_1000
- n_2000

Choose processes
- p_876_n_1000
- p_877_n_1000

Choose sites
- Cyfronet
- TUM

Integration mode
- Integral
- Time derivative

Summary mode
- Aggregate results

OK  Cancel  Help
Measurement specification consists of:

- **Metrics**: what to measure
- Set of objects: where to measure
- Set of partner objects
- Set of program regions

Two main (abstract) base classes:

- **Metrics**: knows how to implement a measurement
- **ActiveMeasurement**: knows how to read it

Many different implementations:

- PMC: sampled, function, aggregated functions, ...
- HLAC: different strategies based on metrics definition
Design: Metrics Specification Language

- Specification must (implicitly) specify the necessary actions
  - when a measurement is defined
  - when a probe is executed
  - when a measurement is read

- Existing languages:
  - ASL (Apart), JavaPSL (U. Vienna): performance properties
  - MDL (U. Madison, Paradyn): low level measurements
  - EARL (Research Center Jülich): trace analysis

- None of them seems to be appropriate for G-PM:
  - different goals and focus
  - in G-PM, language is exposed to user (⇒ must be simple)
Plan: Use a declarative language

```c
my_IO_volume(Process[], procs, File[], file, Region[], reg) {
    vol[p][vt] = IO_volume(p, file, reg) AT end(p, vt)
                 - IO_volume(p, file, reg) AT begin(p, vt);
    result[vt] = SUM(vol[p][vt] WHERE p IN procs);
    return result[];
}
```

- Based data flow principle and unification
- Problem: how to compile / interpret such a language?
Conclusions

- Interactive Grid applications demand
  - on-line performance analysis
  - application specific measurements

- Basic idea:
  - provide set of standard metrics
  - allow for application specific events and data
  - provide specification language to define new metrics

- Simple idea, but many problems to solve
Status

_finished:

- coarse design of monitoring system
- coarse design of G-PM
- specification of GUI and measurement interface

Current and future work:

- first implementation of OCM-G, PMC, UIVC
- demonstrators for HLAC
- design of metrics specification language
- implementation of a compiler/interpreter
- optimization of measurements