Autonomic QoS in ASSIST
Grid-aware components

Marco Aldinucci, Marco Danelutto, Marco Vanneschi
Dept. of Computer Science, University of Pisa, Italy
& ISTI - CNR, Pisa, Italy
Motivating ...
  - high-level programming for the grid
  - application adaptivity for the grid

ASSIST basics & adaptivity in ASSIST
  - mechanisms
  - demo & some experiments

Components & QoS
  - autonomic managers
  - QoS contracts

Concluding remarks
concurrency exploitation, concurrent activities set up, mapping/scheduling, communication/synchronization handling and data allocation, ...

manage resources heterogeneity and unreliability, networks latency and bandwidth unsteadiness, resources topology and availability changes, firewalls, private networks, reservation and jobs schedulers, ...

... and a non trivial QoS for applications
not easy leveraging only on middleware

D. Gannon et al. opened the way (GrADS@Rice)
“moving most of the Grid specific efforts needed while developing high-performance Grid applications from programmers to grid tools and run-time systems”

ASSIST is a high-level programming environment for grid-aware // applications. Developed at Uni. Pisa within several national & EU projects. First version in 2001. Open source under GPL.
app = graph of modules

input

P1

P2

P3

P4

output
app = graph of modules
Programmable, possibly nondeterministic input behaviour

Sequential or parallel module

Typed streams of data items
ASSIST parmod
An “input section” can be programmed in a CSP-like way.
An “input section” can be programmed in a CSP-like way.

Data items can be distributed (scattered, broadcasted, multicasted) to a set of **Virtual Processes** which are named accordingly to a topology.
An “input section” can be programmed in a CSP-like way.

Data items can be distributed (scattered, broadcasted, multicasted) to a set of Virtual Processes which are named accordingly to a topology.

Data items partitions are elaborated by VPs, possibly in an iterative way:

\[ \text{while}(...) \]
\[ \text{forall } \text{VP}(\text{in}, \text{out}) \]
\[ \text{barrier} \]

Data is logically shared by VPs (owner-computes).
An “input section” can be programmed in a CSP-like way.

Data items can be distributed (scattered, broadcasted, multicasted) to a set of Virtual Processes which are named accordingly to a topology.

Data items partitions are elaborated by VPs, possibly in iterative way:

while(...)forall VP(in, out) barrier

data is logically shared by VPs (owner-computes)

Data is eventually gathered accordingly to an user defined way.
An “input section” can be programmed in a CSP-like way.

Data items can be distributed (scattered, broadcasted, multicasted) to a set of **Virtual Processes** which are named accordingly to a topology.

Data items partitions are elaborated by VPs, possibly in an iterative way:

```latex
define VPs
while(...) for all VP(in, out)
  barrier

data is logically shared by VPs (owner-computes)
```

Data is eventually gathered accordingly to an user defined way.

Easy to express standard paradigms (skeltons), such as farm, deal, haloswap, map, apply-to-all, forall, ...
parmod implementation

Virtual Processes

VP manager (VPM)

input manager

processes
Compiling & running

QoS contract

ASSIST program

ASSIST compiler

resource description

XML

executable code

(linux, mac, M$win)
Compiling & running

QoS contract
ASSIST program

ASSIST compiler

resource description XML

executable code (linux, mac, M$win)

Run

Network of processes

Managers
AM+MAMs

Grid execution agent (GEA)

launch

query new resources

reconf commands
Adaptivity aims to dynamically control program configuration (e.g. parallel degree) and mapping

- for performance (high-performance is a natural sub-target)
- for fault-tolerance (enable to cope with unsteadiness of resources, and some kind of faults)
1. Mechanism for adaptivity
   - reconf-safe points
     - in which points a parallel code can be safely reconfigured?
   - reconf-safe point consensus
     - different parallel activities may not proceed in lock-step fashion
   - add/remove/migrate computation & data

2. Managing adaptivity
   - QoS contracts
     - Describing high-level QoS requirement for modules/applications
   - “self-optimizing” modules/components
     - under the control of an autonomic manager
Mechanisms

- At parmod level
  - add/remove/migrate VPs
  - very low-overhead due to knowledge coming from high-level semantics + suitable compiling tools
- At component level
  - create/destroy/wire/unwire parallel entities
  - medium/large overhead due to underlying API for staging, run, ...
- Not addressed in this talk (see references in the paper: Europar 05, ParCo 05, ...), I just show a short demo
adaptivity: a working ex.
adaptivity: a working ex.

1. Gexec(newPE, VPM)
adaptivity: a working ex.

1. Gexec(newPE, VPM)
2. acquire consensus
adaptivity: a working ex.

1. Gexec(newPE, VPM)
2. acquire consensus
3. move VP and data

Only 3. is in the critical path
overhead? (mSecs)

<table>
<thead>
<tr>
<th># of PEs involved</th>
<th>Data-parallel (with shared state)</th>
<th>Farm (without shared state)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>add PEs</td>
<td>remove PEs</td>
</tr>
<tr>
<td>$R_l$ on-barrier</td>
<td>1.2 1.6 2.3</td>
<td>0.8 1.4 3.7</td>
</tr>
<tr>
<td>$R_l$ on-stream-item</td>
<td>4.7 12.0 33.9</td>
<td>3.9 6.5 19.1</td>
</tr>
<tr>
<td>$R_t$</td>
<td>24.4 30.5 36.6</td>
<td>21.2 35.3 43.5</td>
</tr>
</tbody>
</table>

GrADS papers reports overhead in the order of hundreds of seconds (K. Kennedy et al. 2004), this is mainly due to the stop/restart behavior, not to the different running env.
Autonomic Computing

- AC emblematic of a vast hierarchy of self-governing systems, many of which consist of many interacting, self-governing components that in turn comprise a number of interacting, self-governing components at the next level down.
- IBM “invented” it in 2001 (control with self-awareness, from human body autonomic nervous system)
  - self-optimization, self-healing, self-protection, self-configuration = self-management
- control loop, of course, exists from mid of last century
Autonomic behavior

- **Monitor**: collect execution stats: machine load, VPM service time, input/output queues lengths, ...
- **Analyze**: instanciate performance models with monitored data, detect broken contract, in and in the case try to indivituate the problem
- **Plan**: select a (predefined or user defined) strategy to reconvey the contract to valid status. The strategy is actually a list of mechanism to apply.
- **Execute**: leverage on mechanism to apply the plan
Autonomic behavior as been included in NGG2/3 (Next Generation Grid) EU founding recommendation as prerequisite for Grid computing.
ASSIST graphs can be enclosed in components
ASSIST graphs can be enclosed in components
they can be wired one another
ASSIST & components

- ASSIST graphs can be enclosed in components
- they can be wired one another
- they may be used to wrap sequential or parallel code (e.g., MPI)
ASSIST graphs can be enclosed in components
they can be wired one another
they may be used to wrap sequential or parallel code (e.g. MPI)
they can be wired to other legacy components (e.g. CCM)
ASSIST & components

- ASSIST graphs can be enclosed in components
- they can be wired one another
- they may be used to wrap sequential or parallel code (e.g. MPI)
- they can be wired to other legacy components (e.g. CCM)
- currently *native component model*, already converging in the forthcoming GCM (authors involved in CoreGRID NoE, WP3)
managed components

- modules and components are controlled by managers
managed components

- modules and components are controlled by managers
- managers implements NF-ports
managed components

- modules and components are controlled by managers
- managers implements NF-ports
- the distributed coordination of managers enable the managing of the application as whole (the top manager being the Application Manager)
## QoS contract
(of the experiment I’ll show you in a minute)

<table>
<thead>
<tr>
<th>Perf. features</th>
<th>QLᵢ (input queue level), QLo (input queue level), TISM (ISM service time), TOSM (OSM service time), Nᵦ (number of VPMs), Tw[i] (VPMᵢ avg. service time), Tₚ (parmod avg. service time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perf. model</td>
<td>Tₚ = max{TISM, ∑ₙ₌₁ Tw[i]/ₙ, TOSM}, Tₚ &lt; K (goal)</td>
</tr>
<tr>
<td>Deployment</td>
<td>arch = (i686-pc-linux-gnu ∨ powerpc-apple-darwin*)</td>
</tr>
<tr>
<td>Adapt. policy</td>
<td>goal_based</td>
</tr>
</tbody>
</table>
Experimenting heterogeneity

Not only Intel+linux: similar experiments has been run on Linux, Mac, Win, and a mixture of them.
Data-par experiment (STP)

Distribution of load among platforms (n. of VPs)

Relative Unbalance

Iteration time

Time (iteration no.)
Data-par experiment (STP)

Distribution of load among platforms (n. of VPs)

Relative Unbalance

Iteration time

Time (iteration no.)
Data-par experiment (STP)

Distribution of load among platforms (n. of VPs)

Relative Unbalance

Iteration time

Time (iteration no.)
Data-par experiment (STP)

Distribution of load among platforms (n. of VPs)

Relative Unbalance

Iteration time
Data-par experiment (STP)

Distribution of load among platforms (n. of VPs)

Relative Unbalance

Iteration time

Time (iteration no.)
Data-par experiment (STP)

Distribution of load among platforms (n. of VPs)

Relative Unbalance

Iteration time

Time (iteration no.)
Conclusions 1/2

- Application adaptivity in ASSIST
  - complex, but transparent (no burden for the programmers)
    - they should just define their QoS requirements
    - QoS models are automatically generated from program structure (and don’t depend on seq. funct.)
  - dynamically controlled, efficiently managed
    - catch both platforms unsteadiness and code irregular behavior in running time
    - performance models not critical, reconfiguration does not stop the application
    - key feature for the grid
Conclusions 2/2

- ASSIST cope with
  - grid platform unsteadiness
  - interoperability with standards
    - and rely on them for many features
  - high-performance
  - app deployment problems on grid
    - private networks, job schedulers, firewalls, ...
  - QoS of the whole application through hierarchy of managers
The work already evolved (paper dates back 8 months)

- "self-optimizing" higher-order components (farm and DAG), e.g. farms of MPI applications semi-automatically wrapped into components

- fault-tolerance support is ongoing

Foundations of QoS and manager hierarchies

- set of interesting proprieties for Grid (FT, performance)

- suitable formal tools to describe contracts for Grid: how describe a contract, how join/split contracts, ..

- in cooperation with many coreGRID partners, new cooperations are welcome ...
Thank you

ASSIST is open source under GPL

http://www.di.unipi.it/Assist.html