Toward autonomic QoS in Grid-aware applications: the ASSIST experiment

Marco Aldinucci
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

Programmable, possibly nondeterministic input behaviour

Sequential or parallel module

Typed streams of data items

P1

P2

P3

P4

input

output
native + standards

ASSIST native or wrap (MPI, CORBA, CCM, WS)

TCP/IP, Globus, IIOP CORBA, HTTP/SOAP
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 iterative way:

```plaintext
while(...) 
    forall VP(in, out) 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 iterative way:

```plaintext
while(...) 
  forall 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

input manager

VP manager (VPM)

VP

VP

VP

VP

VP manager (VPM)

input manager

processes

VP Virtual Processes
Compiling & running

QoS contract

ASSIST compiler

resource description

XML

Run

executable code

(linux, mac, M$win)

Managers

AM+MAMs

Grid execution agent (GEA)

query new resources

launch

reconf commands

Network of processes
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
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>1 → 2</td>
<td>2 → 4</td>
<td>4 → 8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 → 1</td>
</tr>
<tr>
<td>$R_l$ on-barrier</td>
<td>1.2</td>
<td>1.6</td>
</tr>
<tr>
<td>$R_l$ on-stream-item</td>
<td>4.7</td>
<td>12.0</td>
</tr>
<tr>
<td>$R_t$</td>
<td>24.4</td>
<td>30.5</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.
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

Managed element (module, component)

Monitor

Analyze

Plan

Execute

QoS data

broken contract

next configuration

- **monitor**: collect execution stats: machine load, VPM service time, input/output queues lengths, ...
- **analyze**: instantiate performance models with monitored data, detect broken contract, in and in the case try to individuate 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

- **monitor**: collect execution stats: machine load, VPM service time, input/output queues lengths, ...
- **analyze**: instantiate performance models with monitored data, detect broken contract, instantiate 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

**Managed element (module, component)**

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 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 & 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)
ASSIST graphs can be enclosed in components
they can be wired one another
they may 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
- managers implement NF-ports
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>$Q_L_i$ (input queue level), $Q_L_o$ (input queue level), $T_{ISM}$ (ISM service time), $T_{OSM}$ (OSM service time), $N_w$ (number of VPMs), $T_w[i]$ (VPM$_i$ avg. service time), $T_p$ (parmod avg. service time)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perf. model</td>
<td>$T_p = \max{T_{ISM}, \sum_{i=1}^{n} T_w[i]/n, T_{OSM}}$, $T_p &lt; K$ (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>
**experiment: stateless farm**

- **contract:**
  - keep a given service time
  - contract change along the run
Experimenting heterogeneity

Expected work balance among platforms

Platforms

A  B  C  D

BogoMIPS

P3@868MHz  P4@2.5GHz  P4@2GHz  P4@2.8GHz

A  11%
B  30%
C  24%
D  35%
Experimenting heterogeneity

Platforms

A
B
C
D

P3@868MHz
P4@2.5GHz
P4@2GHz
P4@2.8GHz

6000
5500
4500
3000
1500
0

BogoMIPS

Expected work balance among platforms

A
11%
B
30%
C
24%
D
35%

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.)
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
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
Thank you

ASSIST is open source under GPL

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