QoS and Energy Management Coordination using Discrete Controller Synthesis

Noël De Palma    Gwenaël Delaval    Eric Rutten

Grenoble University, INRIA Grenoble

nov. 29, 2010 — GCM 2010, Bangalore
Self-management for green computing

- **green computing** in distributed infrastructures
  grids, clouds, clusters; replication and multi-tiers
  trade-off with more traditional performance issues:
  dependability, scalability
Self-management for green computing

- **green computing** in distributed infrastructures: grids, clouds, clusters; replication and multi-tiers.

Trade-off with more traditional performance issues:
  - dependability, scalability

- Autonomic computing and self-management approach

  - For QoS: acting on admission control, resource provisioning, service degradation

  - For energy: acting on frequency or voltage, state of hardware devices, server consolidation (virtualization)
Control techniques for autonomic computing

- classical control techniques for individual issues, not for coordination
Control techniques for autonomic computing

- classical control techniques
  for individual issues, not for coordination
- discrete controller synthesis (DCS): off-line computation
  from: a behavior model (FSM)
  controllables variables (degrees of freedom)
  a temporal property
  of a controller that, combined with the behavior, will
  enforce the objective, whatever the uncontrollable inputs
Control techniques for autonomic computing

- classical control techniques
  for individual issues, not for coordination
- discrete controller synthesis (DCS): off-line computation
  from: a behavior model (FSM)
  controllables variables (degrees of freedom)
  a temporal property
  of a controller that, combined with the behavior, will
  enforce the objective, whatever the uncontrollable inputs

advantages:

1. automatic generation of coordination manager
2. ease of evolution in strategy
3. correct by construction
Control techniques for autonomic computing

- classical control techniques for individual issues, not for coordination
- discrete controller synthesis (DCS): off-line computation from: a behavior model (FSM)
  controllables variables (degrees of freedom)
  a temporal property of a controller that, combined with the behavior, will enforce the objective, whatever the uncontrollable inputs

advantages:

1. automatic generation of coordination manager
2. ease of evolution in strategy
3. correct by construction

contributions:
applying DCS to model and solve a problem of autonomic administration, of QoS and energy
Reactive systems and automata

- modelling formalism and programming language
  reaction to input flows → output flows
  - data-flow nodes and equations
  - mode automata (FSM)
  - parallel and hierarchical composition

  *synchronous languages, (25+ years)*

  tools: compilers (e.g., Heptagon), code generation, verification, ...
Reactive systems and automata

- modelling formalism **and** programming language
  reaction to input flows → output flows
  - data-flow nodes and equations
  - mode automata (FSM)
  - parallel and hierarchical composition

synchronous languages, (25+ years)

tools: compilers (e.g., Heptagon), code generation, verification, ...

**example:** computing task control, delayable

```
node delayable(r,c,e:bool) returns (a,s:bool)
let automaton
state Idle do
  a = false; s = r and c
  until r and c then Active
  | r and not c then Wait
state Wait do a = false; s = c
  until c then Active
state Active do a = true; s=false
  until e then Idle
end tel
```
Discrete controller synthesis: principle

Goal

Enforcing a temporal property $\Phi$ on a system (on which $\Phi$ does not a priori hold)
Discrete controller synthesis: principle

Goal

Enforcing a temporal property $\Phi$ on a system (on which $\Phi$ does not a priori hold)

Principle (on implicit equational representation)

- **State**: memory
- **Trans**: transition function
- **Out**: output function

Diagram:

```
+----------------+   +----------------+
|   Trans        | -> |   State         |
+----------------+   +----------------+
              ^   ^              
              |   |              
              Y   Z
```

Computation of a controller such that the controlled system satisfies $\Phi$. DCS tool: Sigali (H. Marchand et al.)
Discrete controller synthesis: principle

**Goal**

*Enforcing* a temporal property $\Phi$ on a system (on which $\Phi$ does not a priori hold)

**Principle (on implicit equational representation)**

- *State* memory
- *Trans* transition function
- *Out* output function

- Partition of inputs into controllable ($Y^c$) and uncontrollable ($Y^u$) inputs
Discrete controller synthesis: principle

Goal

Enforcing a temporal property $\Phi$ on a system (on which $\Phi$ does not a priori hold)

Principle (on implicit equational representation)

- **State** memory
- **Trans** transition function
- **Out** output function

- Partition of inputs into controllable ($Y^c$) and uncontrollable ($Y^u$) inputs
- Computation of a controller such that the controlled system satisfies $\Phi$

DCS tool: Sigali (H. Marchand e.a.)
BZR: contracts and DCS

\[
f(x_1, \ldots, x_n) = (y_1, \ldots, y_p)
\]

assume \( e_A \)
enforce \( e_G \)

with \( c_1, \ldots, c_q \)

\[
y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q)
\]
\[
\ldots
\]
\[
y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q)
\]

- built on top of nodes in Heptagon (M. Pouzet e.a.)
- to each contract, associate controllable additional variables, local to the component
**BZR: contracts and DCS**

\[ f(x_1, \ldots, x_n) = (y_1, \ldots, y_p) \]

- Assume \( e_A \)
- Enforce \( e_G \)
- With \( c_1, \ldots, c_q \)

\[ y_1 = f_1(x_1, \ldots, x_n, c_1, \ldots, c_q) \]
\[ \ldots \]
\[ y_p = f_p(x_1, \ldots, x_n, c_1, \ldots, c_q) \]

- Built on top of nodes in Heptagon (M. Pouzet e.a.)
- To each contract, associate controllable additional variables, local to the component
- At compile-time (user-friendly DCS), compute a controller for each component
BZR: contracts and DCS

\[ f(x_1, \ldots, x_n) = (y_1, \ldots, y_p) \]

**Motivation**

Reactive systems and DCS

QoS and energy management

Synchronous controller design

Conclusion

- built on top of nodes in Heptagon (M. Pouzet e.a.)
- to each contract, associate controllable additional variables, local to the component
- at compile-time (user-friendly DCS), compute a controller for each component
- when no controllables: verification by model-checking
BZR example

mutual exclusion enforced by DCS in BZR

- two instances of the delayable node
- declaration of $c_1$ and $c_2$ as controllable variables
- simple contract:

<table>
<thead>
<tr>
<th>twotasks($r_1,e_1,r_2,e_2$) = $a_1,s_1,a_2,s_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>enforce not ($a_1$ and $a_2$)</td>
</tr>
<tr>
<td>with $c_1,c_2$</td>
</tr>
</tbody>
</table>

(a$_1$, s$_1$) = delayable($r_1,c_1,e_1$)
(a$_2$, s$_2$) = delayable($r_2,c_2,e_2$)

some requests $r_i$ are blocked, and memorized
Implementation

- BZR specification
  - contracts, automata
- synchronous compiler
  - hec
- transition system + objectives
- DCS tool
  - sigali
- controller
  - (constraint)
  - triangularize transl. to eq.
  - resolver code gen.
    - hectr1
  - on-line resolver (C)
- sequential code (C, Java, ...)
- link edition
QoS and energy management

application to internet servers
standard pattern of scalability and availability

- different tiers e.g., web, application, database servers
application to internet servers
standard pattern of scalability and availability

- different tiers e.g., web, application, database servers

- servers replicated at deployment time
distribution of requests by load balancer (round-robin)
QoS and energy management

application to internet servers
standard pattern of scalability and availability

- different tiers e.g., web, application, database servers

- servers replicated at deployment time
  distribution of requests by load balancer (round-robin)

- dynamical adaptation:
  - number of requests accepted: maintain QoS, avoid trashing
  - number of replicas: allocate (load) or free (energy) machines
load balancing scheme in a cluster of Web servers

including an admission control:

- accepts/rejects new client requests
to maintain high system throughput
Admission control

load balancing scheme in a cluster of Web servers including an admission control:

- accepts/rejects new client requests to maintain high system throughput
- computes an average CPU load: CPU_AVG
load balancing scheme in a cluster of Web servers including an admission control:

- accepts/rejects new client requests to maintain high system throughput
- computes an average CPU load: CPU_AVG
- uses cost associated to each request type
Admission control

load balancing scheme in a cluster of Web servers including an admission control:

- accepts/rejects new client requests to maintain high system throughput
- computes an average CPU load: CPU_AVG
- uses cost associated to each request type
- important parameter: maximal CPU usage MaxCPU_AC above: request rejected
• dynamically adapting the degree of replication according to the load the system receives
• turn off cluster nodes to save power under lighter load
Energy control

- dynamically adapting the degree of replication according to the load the system receives
- turn off cluster nodes to save power under lighter load

- actuators:
  - increase/decrease the number of server replicas;
  - update the load balancer with new configuration
dynamically adapting the degree of replication according to the load the system receives

- turn off cluster nodes to save power under lighter load

- actuators:
  - increase/decrease the number of server replicas;
  - update the loadbalancer with new configuration

- based on thresholds: MaxCPU_Prov and MinCPU_Prov
Controller coordination

both controllers are developed independently

- without coordination:
  - admission control prevents energy control to add new node
    - higher number of rejected requests
Controller coordination

both controllers are developed independently

- without coordination:
  admission control prevents energy control to add new node
  \[ \rightarrow \] higher number of rejected requests

- we want to ensure QoS onto the minimum of machines
  with minimum rejected requests
both controllers are developed independently

- without coordination:
  - admission control prevents energy control to add new node → higher number of rejected requests
  - we want to ensure QoS onto the minimum of machines with minimum rejected requests
  - we need the controllers to provide a on/off switch used by super-controller

```
prov up
Prov on AC off
Not(min_node)
prov up/down
min_node
Prov on AC off

max_node and CPU_avg > CPU_Max_prov
CPU_avg < CPU_min_prov
prov up
prov timer
AC
Prov off AC on
AC
Prov off AC on
Set timer 4sec

Not (max_node) and CPU_avg > CPU_Max_prov

Timer end
```
Synchronous controller design

for ease of design, correction, and modifiability:

- first describe controllers independently, with on/off switches
- then coordinate with policy to be enforced
Synchronous controller design

for ease of design, correction, and modifiability:
- first describe controllers independently, with on/off switches
- then coordinate with policy to be enforced

automaton for the admission controller (controllable switch $c$)

```
Idle \[\text{act}_{ac} = \text{false}\]

\[\text{not } c\]

Active \[\text{act}_{ac} = \text{true}\]
```
Synchronous controller design: provisioning policy

automaton for the provisioning policy

(only uncontrollable switches)

act_prov_up = true
act_prov_down = false

CPU_avg > CPU_Max_prov / add_machine

CPU_avg < CPU_Min_prov / remove_machine

act_prov_up = false
act_prov_down = true

timer_end
and not max_node

(CPU_avg < CPU_Min_prov)
and not min_node / remove_machine

(CPU_avg > CPU_Max_prov)
/ add_machine

CPU_avg > CPU_Max_prov / add_machine

CPU_avg < CPU_Min_prov / remove_machine

act_prov_up = true
act_prov_down = true

timer_end
and not max_node

act_prov_up = false
act_prov_down = false

timer_end
and max_node
Synchronous controller design: coordination

contract upon parallel composition of individual controllers

1 admission control never active when provisioning is
2 admission control active when provisioning idle

```plaintext
main(...) = ...
enforce (act_ac and not act_prov_up) or (not act_ac and act_prov_up)
with c
```

- **Up**
  - act_prov_up = true
  - act_prov_down = false
  - CPU_avg < CPU_Min_prov
  - and min_node / remove_machine
  - CPU_avg > CPU_Max_prov / add_machine

- **Down**
  - act_prov_up = false
  - act_prov_down = true
  - timer_end
  - and not max_node

- **UpDown**
  - act_prov_up = true
  - act_prov_down = true
  - CPU_avg < CPU_Min_prov / remove_machine

- **Adding**
  - act_prov_up = false
  - act_prov_down = false
  - timer_end
  - and max_node

- **Idle**
  - act_ac = false
  - not c

- **Active**
  - act_ac = true
  - c
Controller Simulation

- Step 4: `provisioning_up` ($\text{CPU\_avg} > \text{CPU\_Max\_prov}$)
  - $\rightarrow$ `add_machine`, provisioning off, admission on

- Step 7: `timer_end`: machine up $\rightarrow$ prov. on, adm. off

- Step 11: `provisioning_up` $\rightarrow$ `add_machine`

- Step 16: `timer_end` and `max_node`: only prov.down on, and adm. on
Conclusions & perspectives

- major challenge: consistent, efficient and flexible coexistence between QoS and energy Managers in the same system
Conclusions & perspectives

- major challenge: consistent, efficient and flexible coexistence between QoS and energy Managers in the same system
- approach: synchronous programming and DCS
  - automatic generation of the controller for cooperation of multiple autonomic managers from high-level policy,
  - ease of evolution of the coordination strategy
  - correctness by construction of the generated controller
Conclusions & perspectives

- major challenge: consistent, efficient and flexible coexistence between QoS and energy Managers in the same system

- approach: synchronous programming and DCS
  - automatic generation of the controller for cooperation of multiple autonomic managers from high-level policy,
  - ease of evolution of the coordination strategy
  - correctness by construction of the generated controller

- perspectives
  - integrate new consolidation manager based on virtual machines and another QoS manager based on service degradation
  - further managers: repair, network driver management, ...
  - coordination across tiers ...
  - more elaborate control: weights and optimization, ...