Coordinating System Administration Loops using Reactive and Synchronous Models

Soguy Mak-Karé Gueye

- SARDES -
- INRIA Grenoble/ LIG -

Synchron, November 2011

Supervisors: Eric Rutten - Noël de Palma
Outline

1. Motivation
2. Our approach
3. Coordinating two energy-aware managers
4. Experimental evaluation
5. Conclusion
Outline

1 Motivation
   - Autonomic computing
   - Automation of well-identified management tasks
   - Requirement for complete system self-management

2 Our approach

3 Coordinating two energy-aware managers

4 Experimental evaluation

5 Conclusion
Autonomic computing

Computing Systems
- Distributed systems involving many nodes
- Heterogeneous and dynamic environment
- Whsufficient hand administration

Autonomic Computing: Objectives
- Self-management capabilities for computing systems
  - Self-Optimization
  - Self-Healing
  - Self-Configuration
  - Self-Protection
Automation of well-identified management tasks

Objectives
- Less errors
- Higher reactivity
- Better usage of resources

Infrastructures for Performance management
- Oceano
- Cluster Reserves

Infrastructures for Availability management
- Rainbow
- JAGR
### Requirement for complete system self-management

#### Use of several Autonomic Managers
- Multiple autonomic managers have to co-exist in the same system
- Need for coordination to avoid:
  - Conflicting decision
  - System inconsistency

#### ad-Hoc Infrastructures
- Architecture for Autonomic Management coordination
- Architecture for Coordinating Multiple Self-Management Systems
Outline

1 Motivation

2 Our approach
   - Techniques for administration loops coordination
   - Synchronous programming
   - Discrete Controller Synthesis (DCS)
   - BZR programming language

3 Coordinating two energy-aware managers

4 Experimental evaluation

5 Conclusion
Techniques for administration loops coordination

**Coordination Challenges**
- Synchronizing AMs’ execution
- Logical control of AMs’ operations

**Synchronous Approach**
- Parallelism
- Synchronization
- Determinism
Synchronous programming

- 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

```plaintext
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 (DCS): 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 $\Phi$
**Discrete controller synthesis (DCS): 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 $\Phi$
Discrete controller synthesis (DCS): 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 $\Phi$
Discrete controller synthesis (DCS): 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 programming language [http://bzr.inria.fr/]

- built on top of nodes in Heptagon
- 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 controllable inputs: verification by model-checking

\[ \text{delayable}(r, c, e) = a, s \]

\[ \begin{align*}
\text{Idle} & \quad \text{Wait} \\
\text{Active} & \quad \text{Active}
\end{align*} \]

\[ a = \text{false} \quad \text{and not } c \\
\text{r and c/s} \quad \text{c/s}
\]

- \[ t_{\text{otasks}}(r_1, e_1, r_2, e_2) = a_1, s_1, a_2, s_2 \]
- \[ \text{enforce not } (a_1 \text{ and } a_2) \]
- \[ \text{ith } c_1, c_2 \]

\[ (a_1, s_1) = \text{delayable}(r_1, c_1, e_1) \]
\[ (a_2, s_2) = \text{delayable}(r_2, c_2, e_2) \]
Outline

1 Motivation

2 Our approach

3 Coordinating two energy-aware managers
   - Autonomic computing framework
   - Autonomic managers to be coordinated
     - Autonomic manager: Sizing
     - Autonomic manager: Dvfs
     - Use of both Sizing and Dvfs administration loops
   - Coordination controller design

4 Experimental evaluation

5 Conclusion
TUNE: Features

- Build system with self-management capabilities (even for legacy systems)
- Allows to integrate several autonomic managers to the same system
- Does not coordinate managers’ execution
Sizing

- Ensures good performance while optimizing resource usage.
- Dynamically adapts the number of replicated servers to the load on the system.
**Dvfs**

- Ensures good performance while optimizing the energy consumption.
- Dynamically adapts the CPU frequency/voltage level of server to the load that server receives.

**Diagram:**

- Single node
  - %CPU Average (Moving Average)
    - %CPU Average
      - CPU (Frequency / Voltage)
        - Increase CPU (freq/volt)
        - Decrease CPU (freq/volt)
  - < minimum_threshold && not min_frequency
  - > maximum_threshold && not max_frequency
Use of both Sizing and Dvfs administration loops

Objectives
- Local energy optimization: Dvfs
- Global energy optimization: Sizing

Efficient use of both managers
- Global optimization before Local optimization
  May not be achieved without coordination
Motivation
Our approach
Coordinating two energy-aware managers
Experimental evaluation
Conclusion

Coordination controller design

- Modeling system composed of managers sizing and Dvfs
- Synthesis of Discrete controller
Describes the control of Sizing operations:

- Sizing operations can be allowed/denied according to the value of delay

\[
\text{Delay\_control}(c) = \text{delay}
\]

\[
\begin{array}{c}
\text{Active} \\
\text{Idle}
\end{array}
\]

\[
\begin{align*}
\text{delay} &= \text{false} \\
\text{delay} &= \text{true}
\end{align*}
\]

\[
\begin{align*}
\text{not } c / &\quad \text{c /} \\
\text{Idle} &\quad \text{Active}
\end{align*}
\]
Describes Sizing execution modes
control of upsizing operations
controllable variable: delay

\[
\text{Sizing\_control}(...) = \text{add\_node, remove\_node, adding}
\]
Describes the different states in which the set of Dvfs managers could be:

- **Max**: All CPUs are in highest frequency
- **Min**: All CPUs are in lowest frequency
- **Normal**: Any other case

No ontrollable variables

```
Dvfs_control(minimum, maximum) = min_freq, max_freq
```
allow upsizing operations
when all processors are in their highest frequency level

**Contract**: \((\text{freq\_max AND not delay}) \ OR \ (\text{not freq\_max AND delay})\)
Control Simulation: [with Sim2chro, Verimag]

**step 1**: overload is false and max_freq is false

**step 5**: overload is true but max_freq is false: no upsizing operation

**step 11**: overload is true and max_freq is true: upsizing operation
Outline

1. Motivation
2. Our approach
3. Coordinating two energy-aware managers
4. Experimental evaluation
   - Experimental platform
   - Execution without coordination
   - Execution with coordination
5. Conclusion
Experimental platform

3 Machines: Ubuntu Os
- node 0: 2.17 Ghz, 2.0 Go RAM, Ubuntu-10.4
- node 1: 1.20 Ghz, 1.50 Go RAM, Ubuntu-10.4
- node 2: 1.20 Ghz, 992.3 Mo RAM, Ubuntu-10.4

Network
- Switch 3Com 4300 48PORT

Managed system: 2-tiers architecture
- One Apache server: Load balancer
  Re eives All requests from clients
  forwards them to Tomcat servers
- Replicated Tomcat servers
  treat client’s requests
  degree of replication may vary according to the system load

Use of Jmeter for the simulation of clients' requests

Step 1: increasing load during 2 minutes
Step 2: constant load
Injection of load that is supported at maximum CPW frequency
Without coordination, the same load leads to upsizing operation and the increase of CPW frequency.
With coordination, the same load does not lead to upsizing operation.
Execution with coordination: Injecting higher load

With coordination, upsizing operation is performed only when it is necessary.
<table>
<thead>
<tr>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Motivation</td>
</tr>
<tr>
<td>2 Our approach</td>
</tr>
<tr>
<td>3 Coordinating two energy-aware managers</td>
</tr>
<tr>
<td>4 Experimental evaluation</td>
</tr>
<tr>
<td>5 Conclusion</td>
</tr>
</tbody>
</table>
conclusions & perspectives

- **major challenge**: consistent, efficient and flexible coexistence between autonomic 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,
  - correctness by construction of the generated controller

- **perspectives**
  - large scale coordination with several managers and multi-tiers architecture
  - more elaborated control than mutual exclusion
Conclusions & Perspectives

- Major challenge: consistent, efficient and flexible coexistence between autonomic 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,
  - Correctness by construction of the generated controller

- Perspectives
  - Large scale coordination with several managers and multi-tiers architecture
  - More elaborated control than mutual exclusion
conclusions & perspectives

- major challenge: consistent, efficient and flexible coexistence between autonomic 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,
  - correctness by construction of the generated controller

- perspectives
  - large scale coordination with several managers and multi-tiers architecture
  - more elaborated control than mutual exclusion