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

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
- Unsufficient hand administration

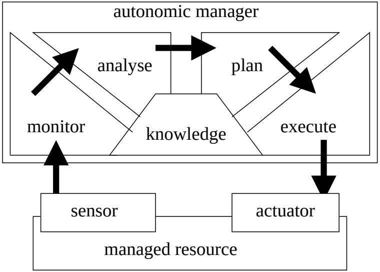
Autonomic Computing: Objectives

- 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

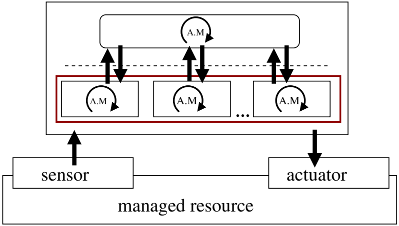
- Multiple autonomic managers have to co-exist in the same system
- Need for coordination to avoid:
  - Conflicting decision
  - System inconsistency

ad-Hoc Infrastructures

- Architecture for Autonomic Management coordination
- Architecture for Coordinating Multiple Self-Management Systems

Outline

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

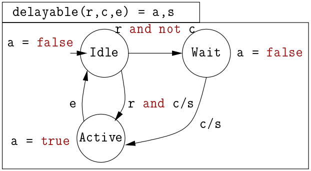
  reaction to input flows → output flows

  - 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




```
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 Φ on a system (on which Φ 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 Φ

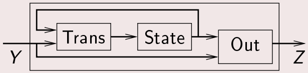
Discrete controller synthesis (DCS): principle

Goal

Enforcing a temporal property Φ on a system (on which Φ 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 Φ

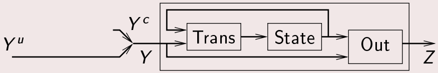
Discrete controller synthesis (DCS): principle

Goal

Enforcing a temporal property Φ on a system (on which Φ 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 Φ

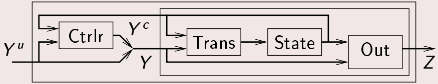
Discrete controller synthesis (DCS): principle

Goal

Enforcing a temporal property Φ on a system (on which Φ does not a priori hold)

Principle (on implicit equational representation)

State	memory
Trans	transition function
Out	output function

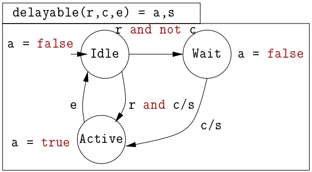


- Partition of inputs into controllable (Y^c) and uncontrollable (Y^u) inputs
- Computation of a controller such that the controlled system satisfies Φ

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



$\texttt{twotasks}(r_1, e_1, r_2, e_2) = a_1, s_1, a_2, s_2$

enforce not $(a_1$ **and** $a_2)$

with c_1, c_2

$$(a_1, s_1) = \texttt{delayable}(r_1, c_1, e_1)$$

$$(a_2, s_2) = \texttt{delayable}(r_2, c_2, e_2)$$

Outline

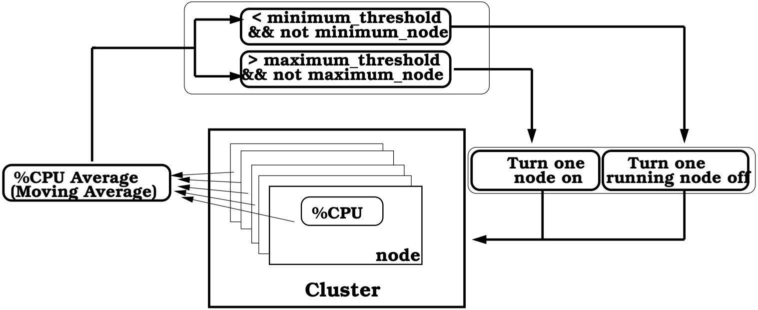
TUNe *[ACM, SAC'08]*

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

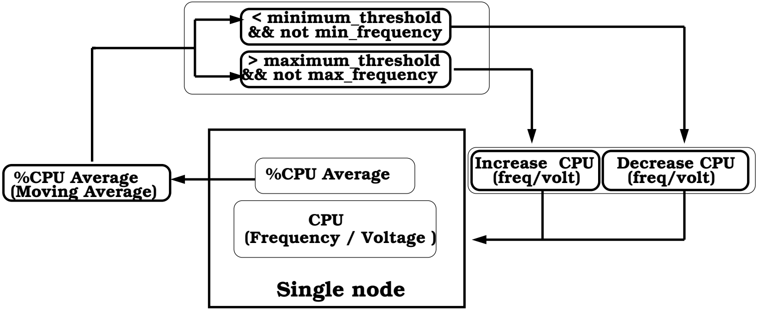
Sizing

- Ensures good performance while optimizing resources 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



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

Coordination controller design

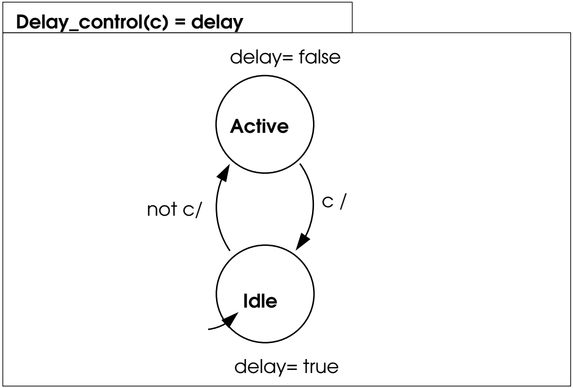
Coordination controller design

- Modeling system composed of managers sizing and Dvfs
- Synthesis of Discrete controller

Modeling Manager Sizing 1/2

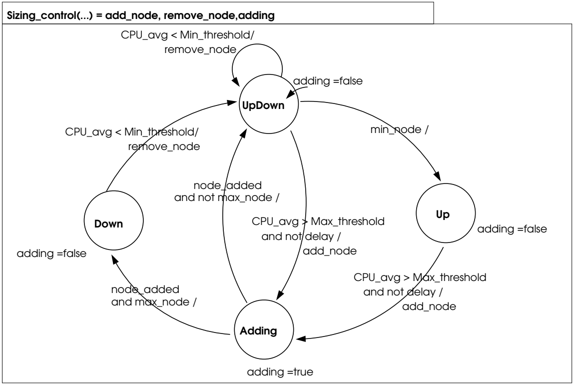
Describes the control of Sizing operations:

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



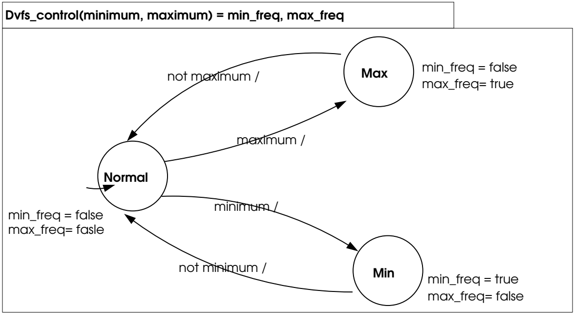
Modeling Manager Sizing 2/2

- Describes Sizing execution modes
  - control of upsizing operations
- controllable variable: **delay**



Modeling Manager Dvfs

- 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 controllable variables



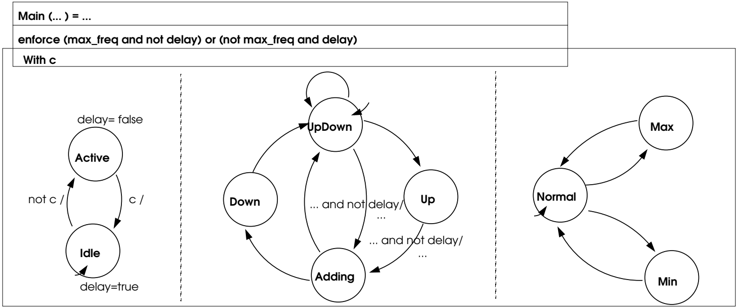
Synchronous control of Sizing and Dvfs

Coordination policies

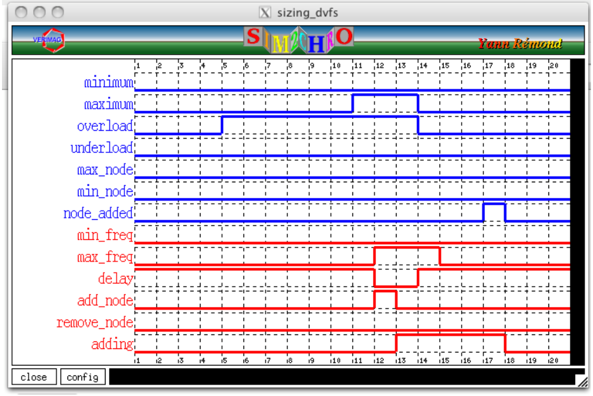
- allow upsizing operations
    - when all processors are in their highest frequency level

Control with

Contract: (freq_max *AND* not delay) *OR* (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

Experimental platform

3 Machines: Ubuntu Os

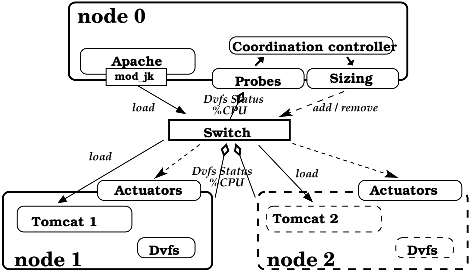
- node 0: 2.17 Ghz, 2.0Go RAM, Ubuntu-10.4
- node 1: 1.20 Ghz, 1.50Go RAM, Ubuntu-10.4
- node 2: 1.20 Ghz, 992.3Mo RAM, Ubuntu-10.4

Network

- Switch 3Com 4300 48PORT

Managed system: 2-tiers architecture

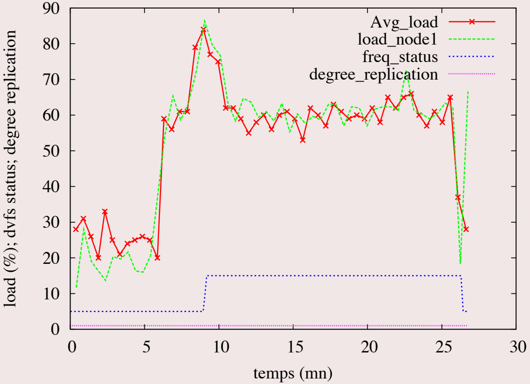
- **One Apache server :** Load balancer
  - Receives All requests from clients
  - forwards them to Tomcat servers
- **Replicated Tomcat servers**
  - treat client's requests
  - degree of replication may vary according to the system load



Use of Jmeter for the simulation of clients' requests

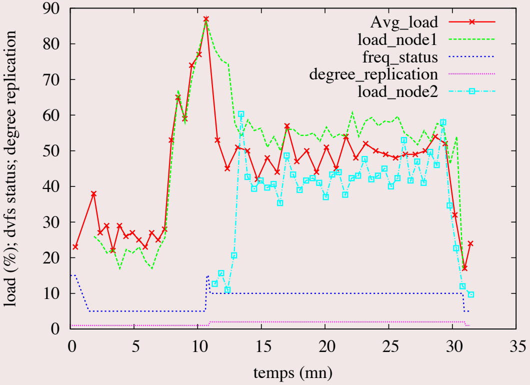
- Step 1: increasing load during 2 minutes
- Step 2: constant load

Sizing operation disabled



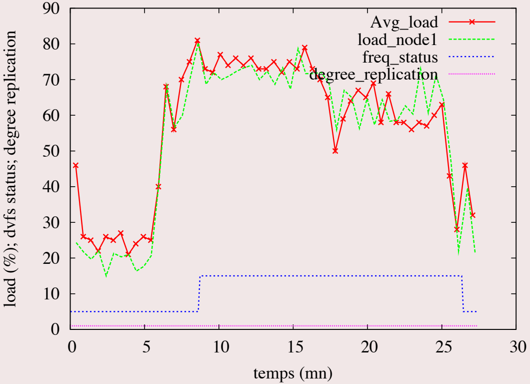
Injection of load that is supported at maximum CPU frequency

Sizing enabled, without coordination



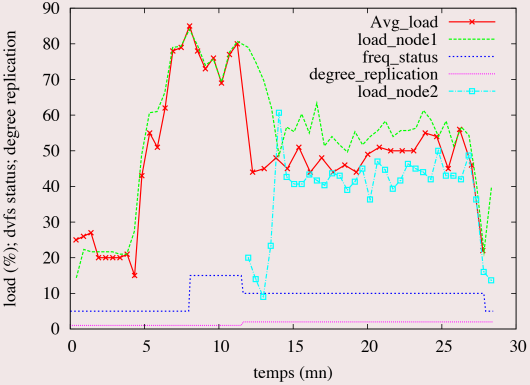
Without coordination, the same load leads to upsizing operation and the increase of CPU frequency

Sizing enabled, with coordination



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.

Outline

conlusions & 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

conlusions & 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

conlusions & 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