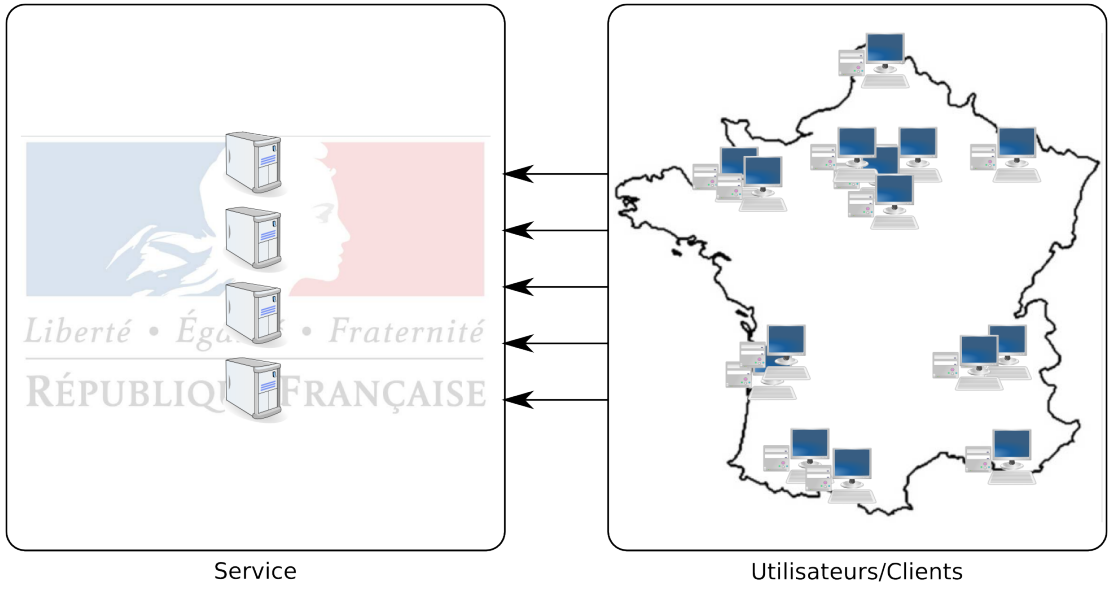
Toward Self-Optimization of Autonomic Systems

CHRISTOPHE TATON

Ph.D Defense
Grenoble Institute of Technology (Grenoble INP)

December 2, 2008

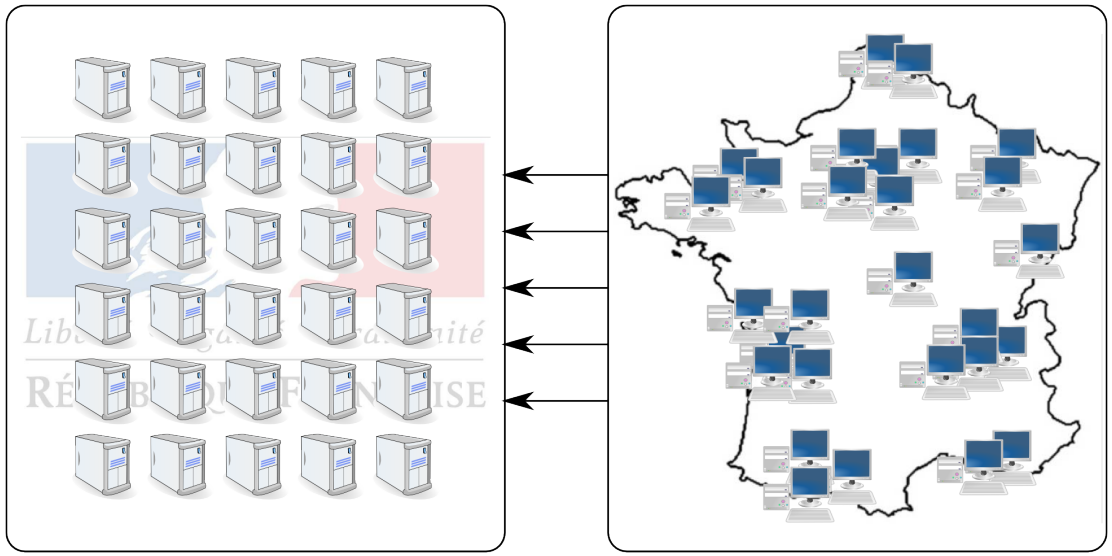
Introduction
French online income declaration service (2005)



- System capacity: 1,5 million users
- Actual load: 3,7 million users

Bad capacity planning:
System overloaded!

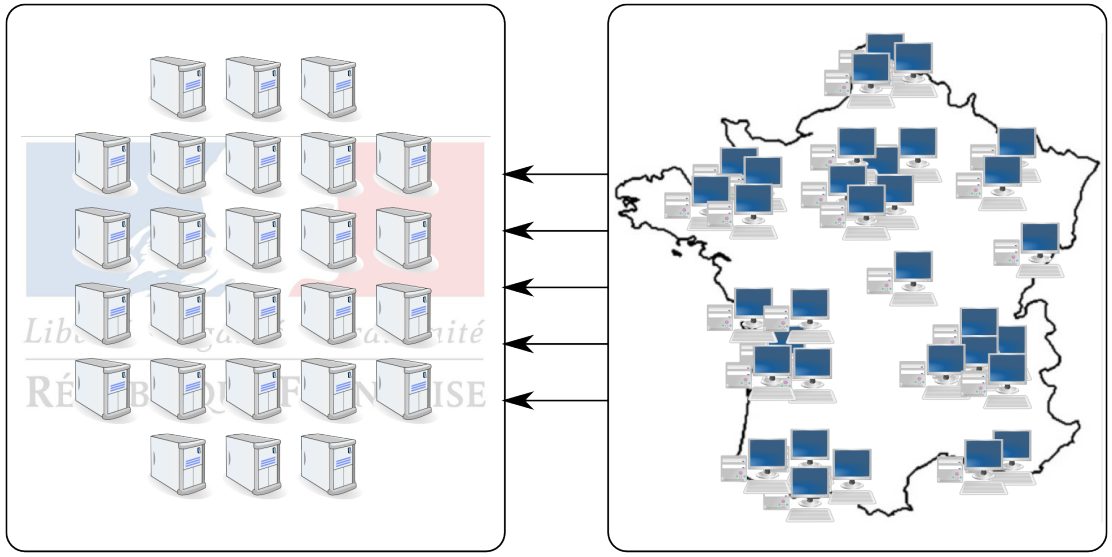
Introduction
French online income declaration service (2006)



- System capacity: 10 million users
- Actual load: 5,7 million users

Sub-optimal capacity planning:
almost 50% resources wasted!

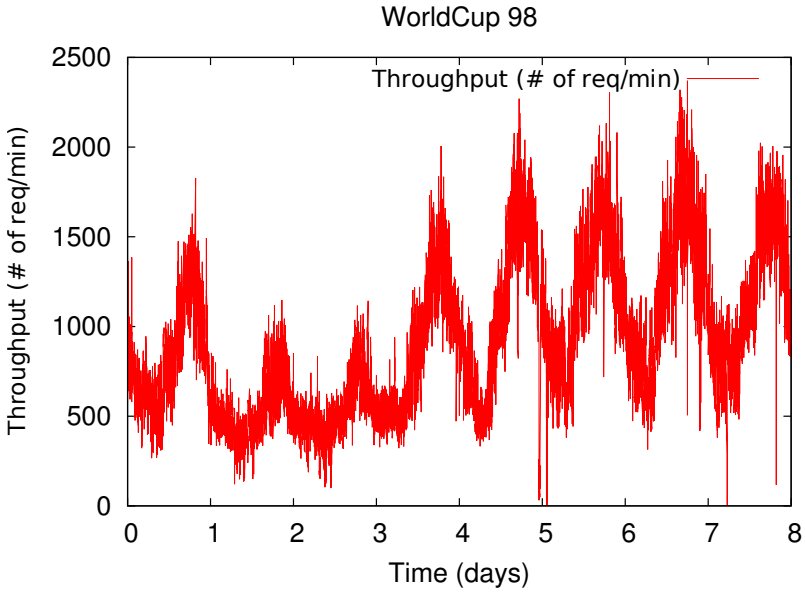
Introduction
French online income declaration service (2007)

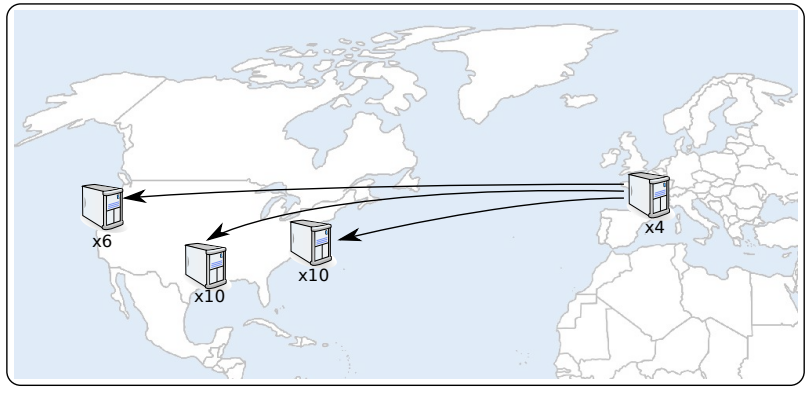


- System capacity: 8 million users
- Actual load: 7.4 million users

Optimal capacity planning!

Introduction
WorldCup'98 Web site





WorldCup 98 Internet Service Architecture

- Dynamic workload: min/max load ratio = $\times 4$
- Fixed system size: 30 servers
  - France Télécom, Hewlett Packard

Challenges
Building self-optimized autonomic systems

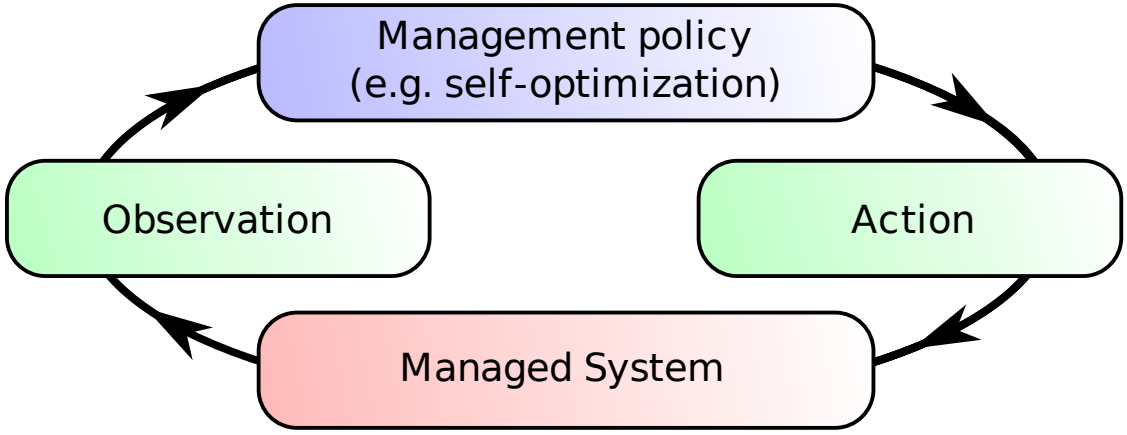
System capacity planning

- Performance and availability guarantees
- Load variation
- Oscillations
- Legacy systems

Dynamic software architecture

- Distributed systems
- Synchronized deployments
- Dynamic architectures

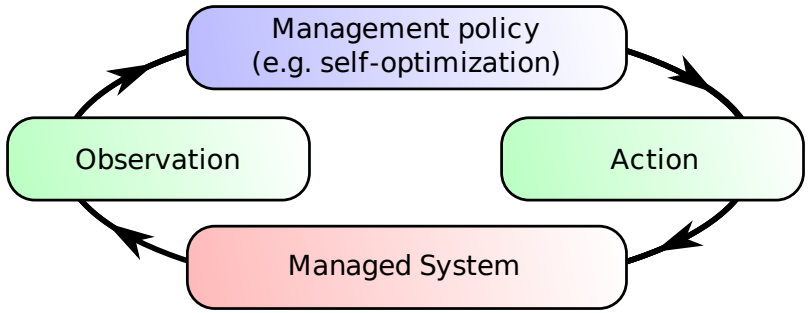
Self-Optimization of Dynamic Software Architecture



Feedback control-loop

Overview

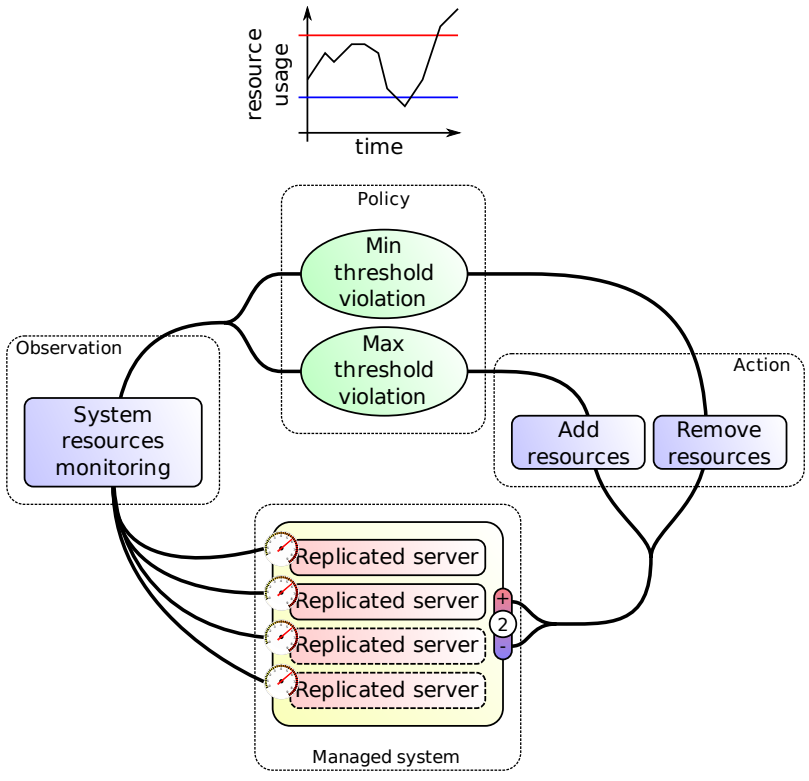
- Contributions
  - Self-optimization policy
  - Actions
  - Observations
- Conclusions & Perspectives



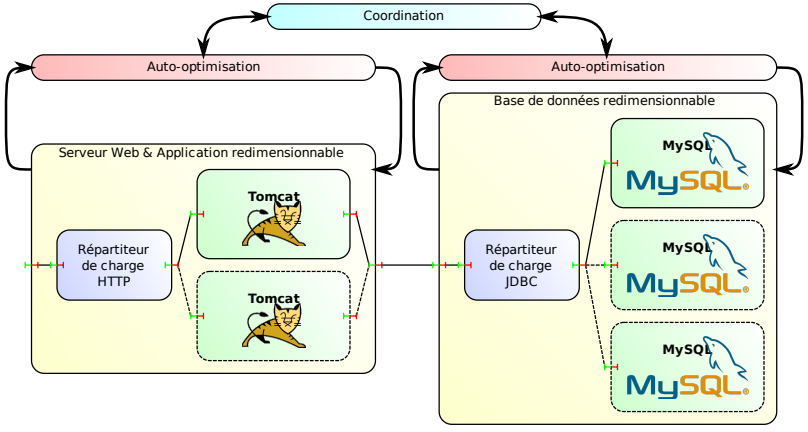
Contributions

Self-optimization policy
Overview

- Feedback control-loop
- Threshold based policy
- Resource usage thresholds
- $\pm n$ resources

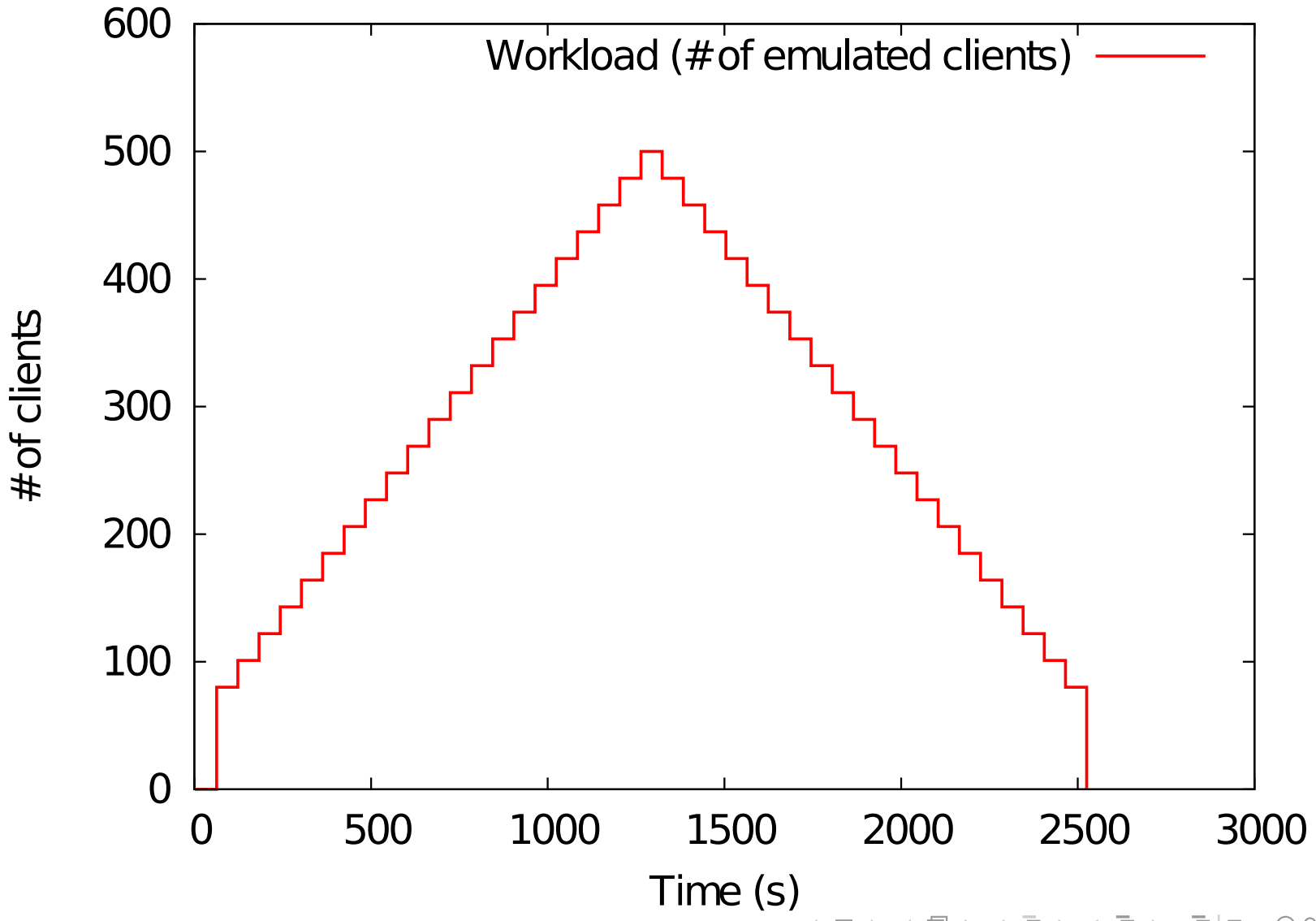


Multi-tier Service Dynamic provisioning

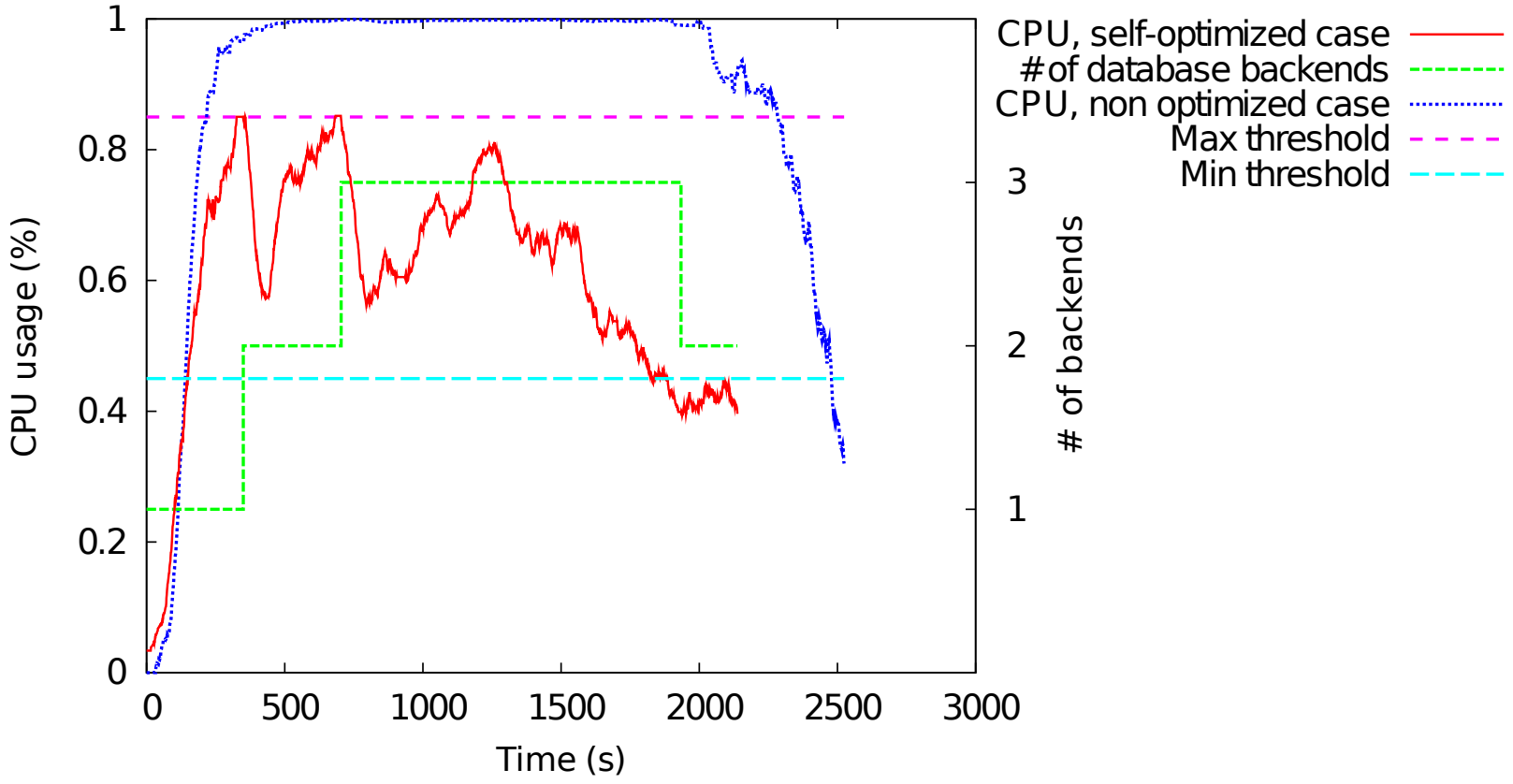


- J2EE System
  - Web & Application: Apache Tomcat v5 (1 or 2 instances)
  - Database: MySQL v5 & Sequoia (from 1 to 3 instances)
- Benchmark: RUBiS (eBay)
  - Gradual load variations

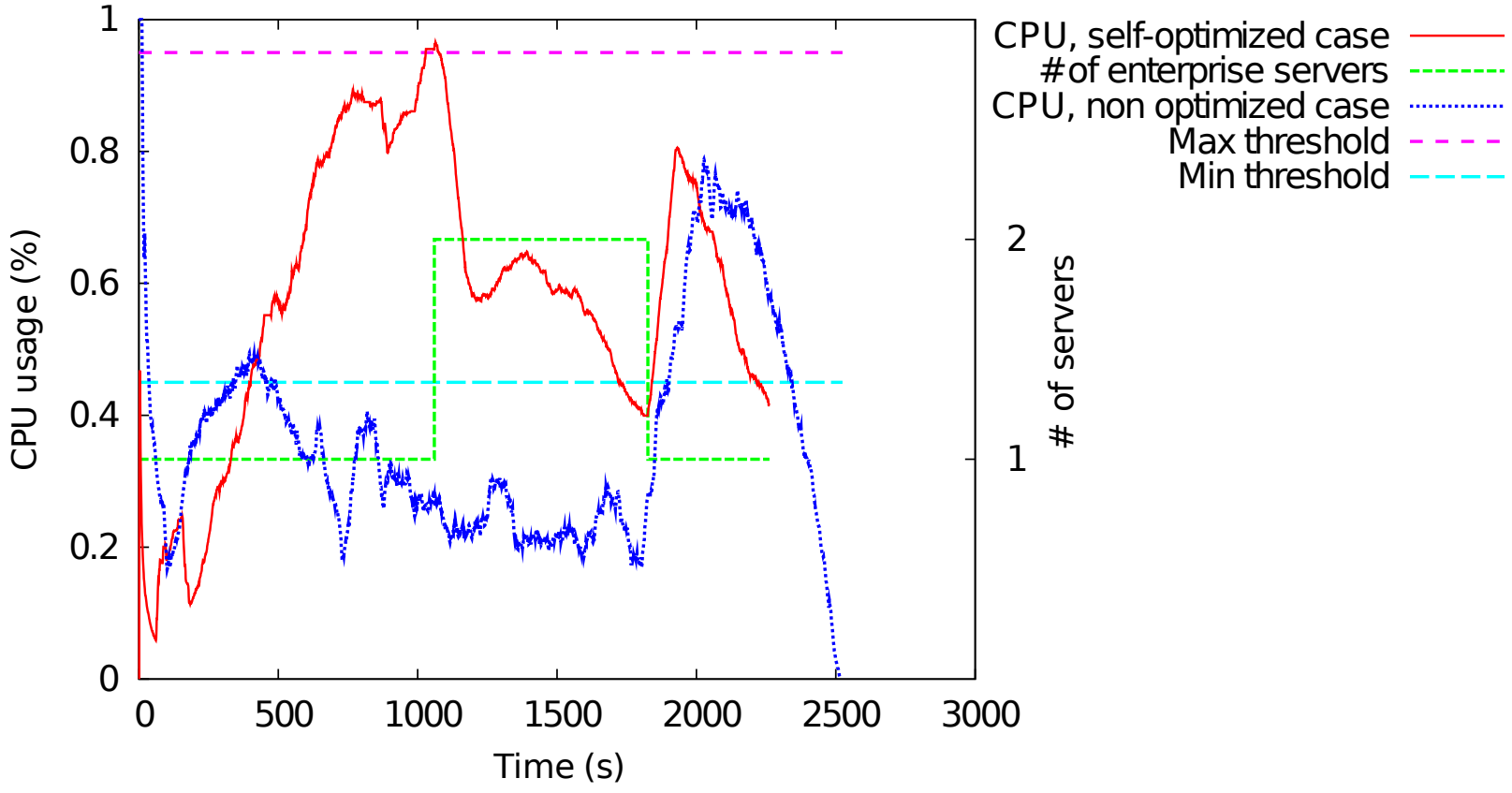
Gradual load variations



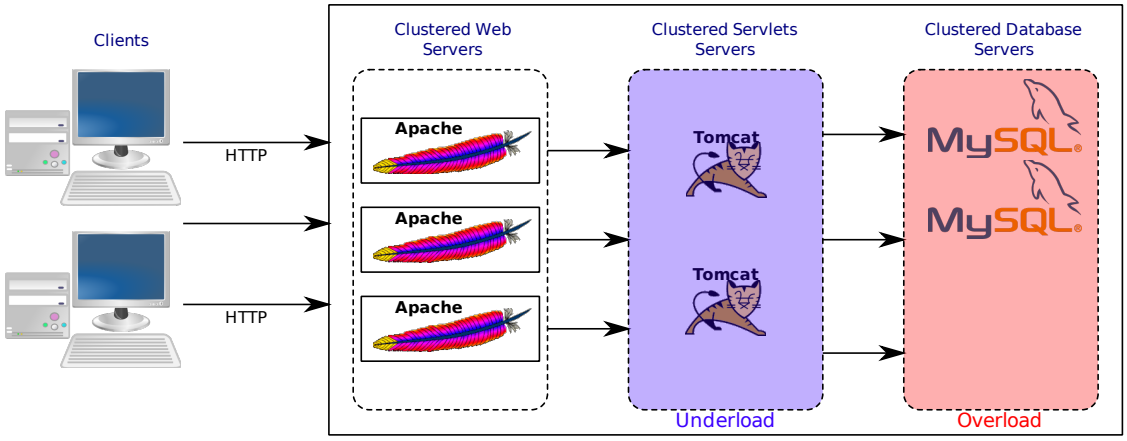
System behaviour
Database servers



System behaviour
Web & Application servers

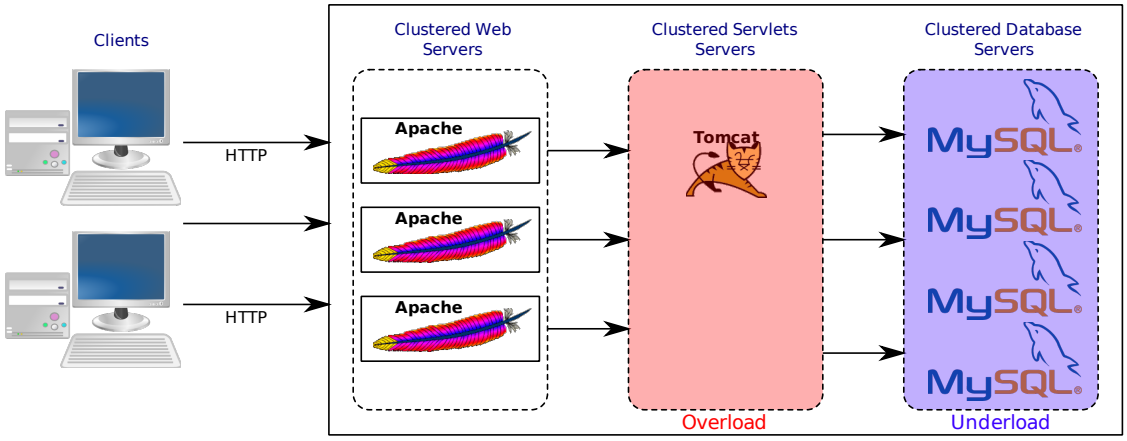


Oscillations
Example (1)



- Sub-system dependencies
  - X directly linked to Y
  - X overload may induce Y underload
- Harmful uncorrelated separate reactions
  - Over/Under-load: add/remove resources

Oscillations
Example (2)



- Sub-system dependencies
  - X directly linked to Y
  - X overload may induce Y underload
- Harmful uncorrelated separate reactions
  - Over/Under-load: add/remove resources

Oscillation prevention
Related work

- Global inhibition delay (Cluster-on-Demand) [Moore et al., 2002]
  - Freeze of the whole system
- Fake resource removal (Chameleon) [Soundararajan et al., 2006]
  - Database specific approach
- Control theory approaches [Hellerstein et al., 2004]
  - Accurate system modeling

Approach toward oscillation prevention

Design principles

- Architecture knowledge
- Sub-system dependencies identification
- Allow coordination
- Reconfiguration inhibition for systems with direct dependencies

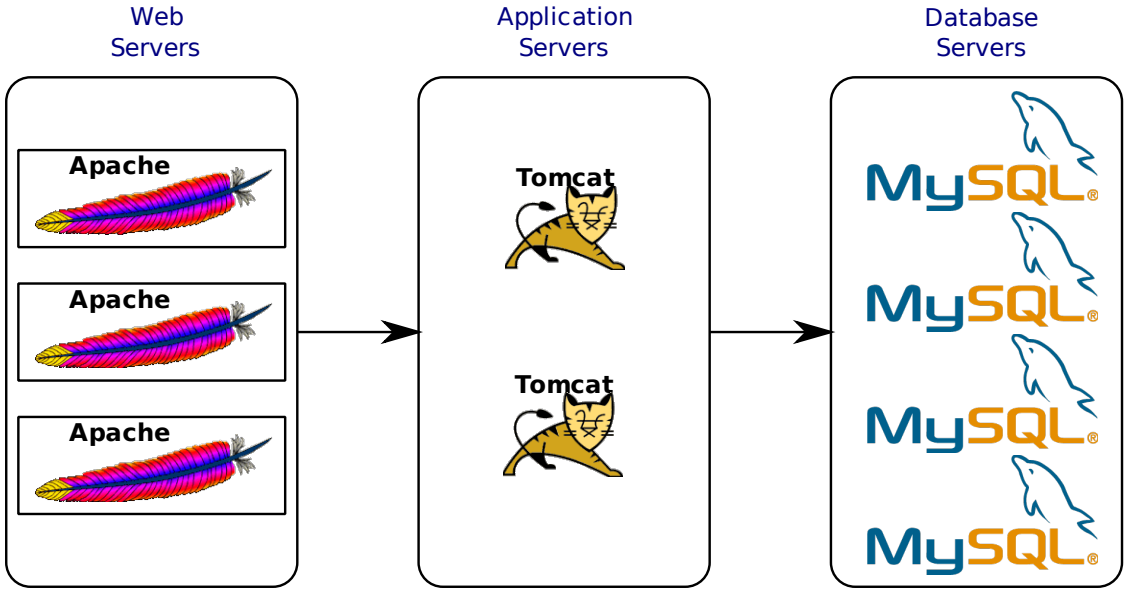
Implementation

- Jade autonomic management platform
- Fractal software components
- Self-optimization manager with oscillation prevention

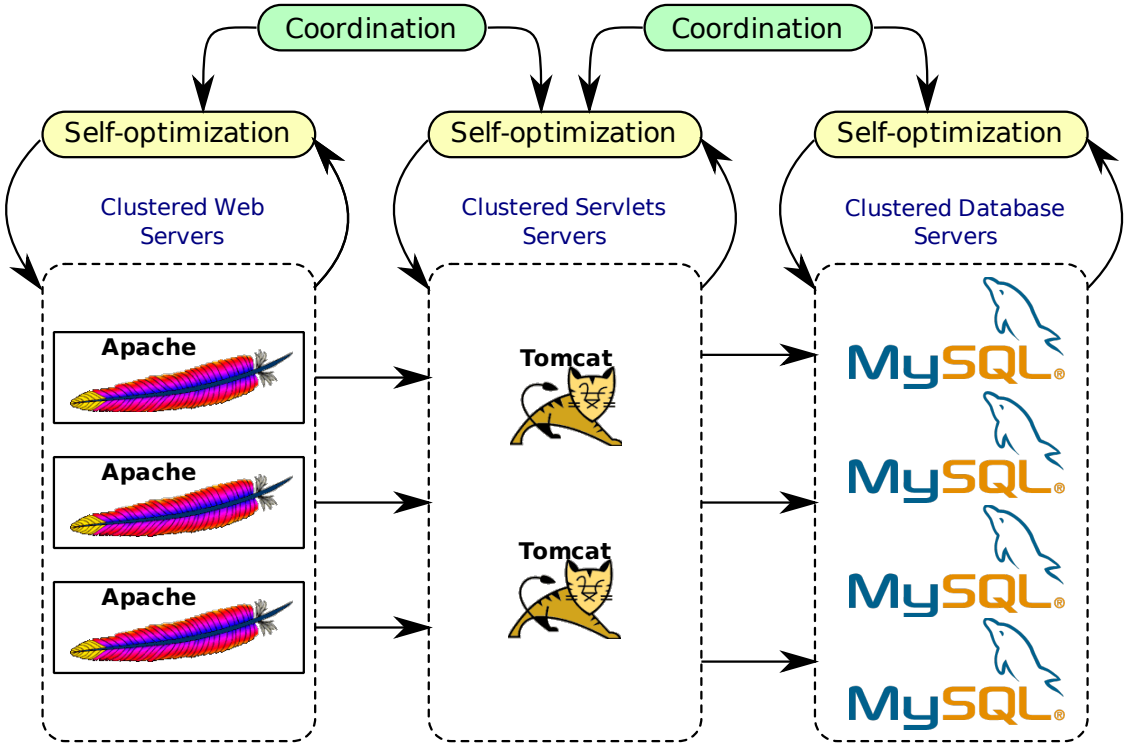
Implementation

Principles

- Components represent sub-systems
- Bindings symbolize dependencies



Implementation
Self-optimized J2EE Internet Service

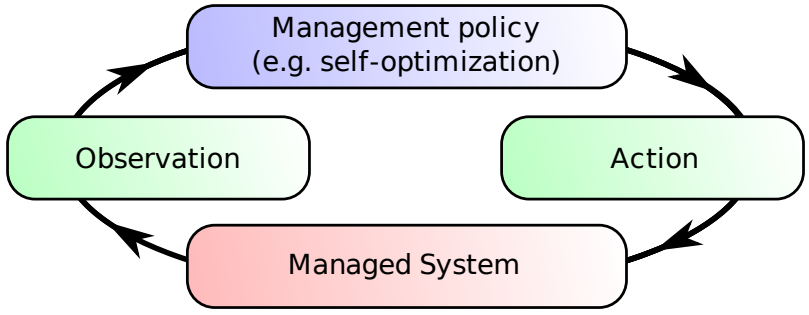


Dynamic provisioning inhibition rule

Resizing on Database servers
inhibits
Resizing on Application and Database servers

Overview

- Contributions
  - Self-optimization policy
  - **Actions**
  - Observations
- Conclusions & Perspectives



Architecture Description Language (ADL)

Architecture Description Language

- System elements: Components

```
<component name="my-component" ...> ... </component>
```

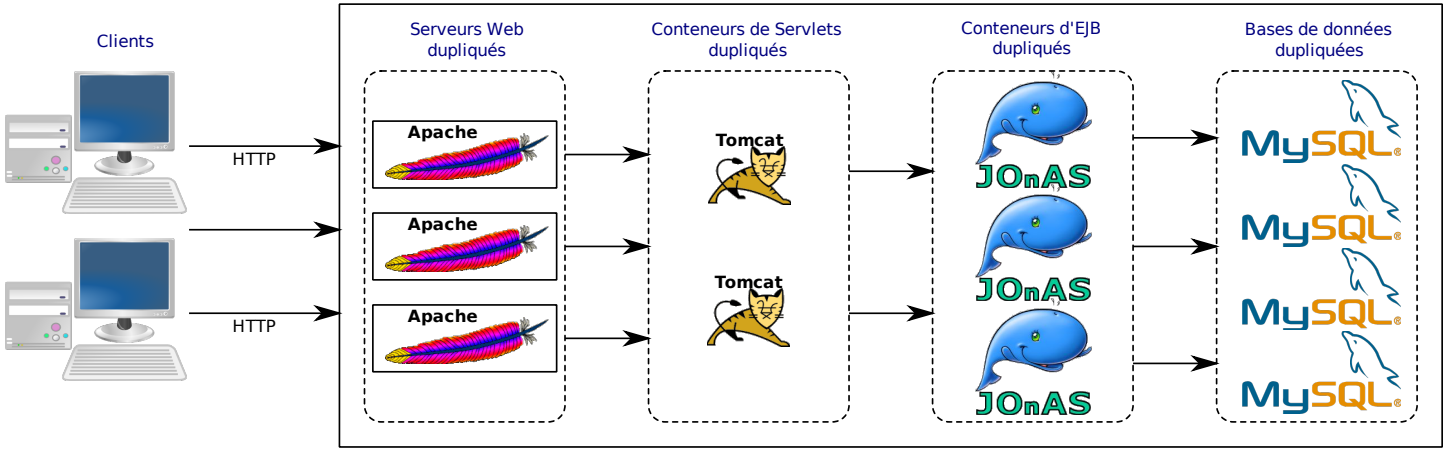
- Sub-system relationships: Interfaces and Bindings

```
<binding name="my-binding"
        client="my-client-itf" server="my-server-itf" .../>
```

Deployment

ADL instantiation process

Distributed deployment synchronization
Example



Distributed deployment of a complex architecture (Multi-tier J2EE)

- What if we parallelize all deployments?
- Invariant: backends started before frontends
- Dependencies through architecture

Related work
Distributed deployment coordination

- Deployment synchronization encoded in factories (Fractal ADL/Julia Factories) [Bruneton et al., 2006]
- Factory parameter (Prism) [Mikic-Rakic et al., 2005]
- Simple ADL parameter (ProActive) [Caromel et al., 2007]
  - Does not allow arbitrary synchronizations
- Generated deployment procedure (IA Planning or Constraints solving) [Arshad, 2006]
  - No control on synchronization
- Workflow languages (SmartFrog, BPEL) [Goldsack, 2003] [Keller et al., 2004]
  - Not applicable to architecture deployments

Approach
Distributed deployment synchronization

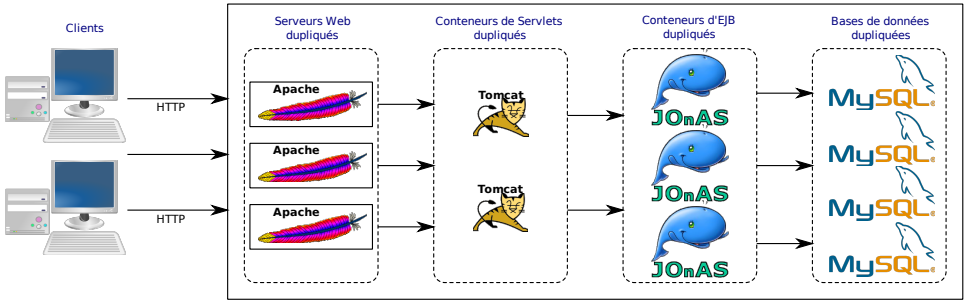
Design principles

- Package
  - Description, deployment and architectural unit
  - 1^{st}-class programming language entity
  - Deployment procedure
- Distributed deployment
  - Coordination and synchronization of local package deployments

Implementation

- Mozart/Oz distributed platform
  - Using Oz synchronization abilities
- FructOz framework
  - Fractal component model

Implementation
Synchronized distributed deployment




```
proc {DeployJ2EE NWebs NServlets NEJBs NDBs}
    {Barrier {MakeList NDBs DeployDB}}
    % Require all DB servers up

    {PartialBarrier {MakeList NEJBs DeployEJB} (NEJBs/2)}
    % Ok when 1/2 EJB servers are up

    {Barrier {MakeList NServlets DeployServlet}}
    % Require all Servlet servers up

    {PartialBarrier {MakeList NWebs DeployWeb} 1}
    % Ready as soon as one Web server instance is up
end
```

Distributed deployment

Database deployment procedure

```
% Set of DB servers initially empty
DBTier = {SNew}

% Deployment procedure for a single DB server
proc {DeployDB}
    Host = {AllocateHost Cluster}
    DB = {RemoteDeploy Host DBPkg}
in
    {DBTier add(DB)}
end
```

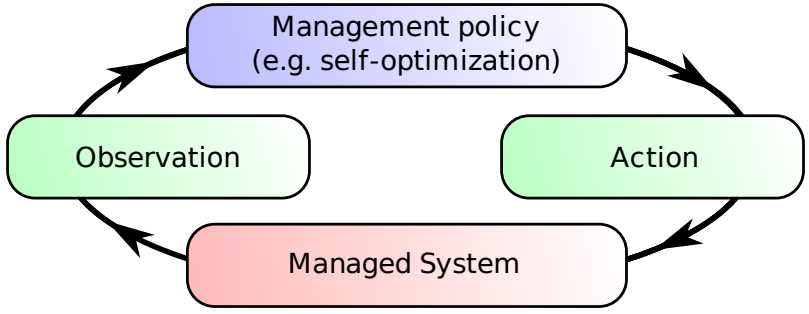
Synchronization barrier

Partial synchronization barrier

```
proc {PartialBarrier LDeploys N}
   Continue % Synchronization signal (undefined yet)
   Counter = {NewCell N} % Counter initialized to N
in
   for Deploy in LDeploys do
      thread % Start all deployment procedures in separate threads
         {Deploy}
         local % Thread−safe counter decrement
            NewCount PreviousCount = {Exchange Counter NewCount}
         in
            if (PreviousCount > 0) then
               NewCount = PreviousCount − 1
               if (NewCount == 0) then
                  Continue = true % Synchronization signal when counter reaches 0
               end
            end
         end
      end
   end
   {Wait Continue} % Wait until synchronization signal is defined
end
```

Overview

- Contributions
  - Self-optimization policy
  - Actions
  - **Observations**
- Conclusions & Perspectives



Dynamic architecture instrumentation
Overview

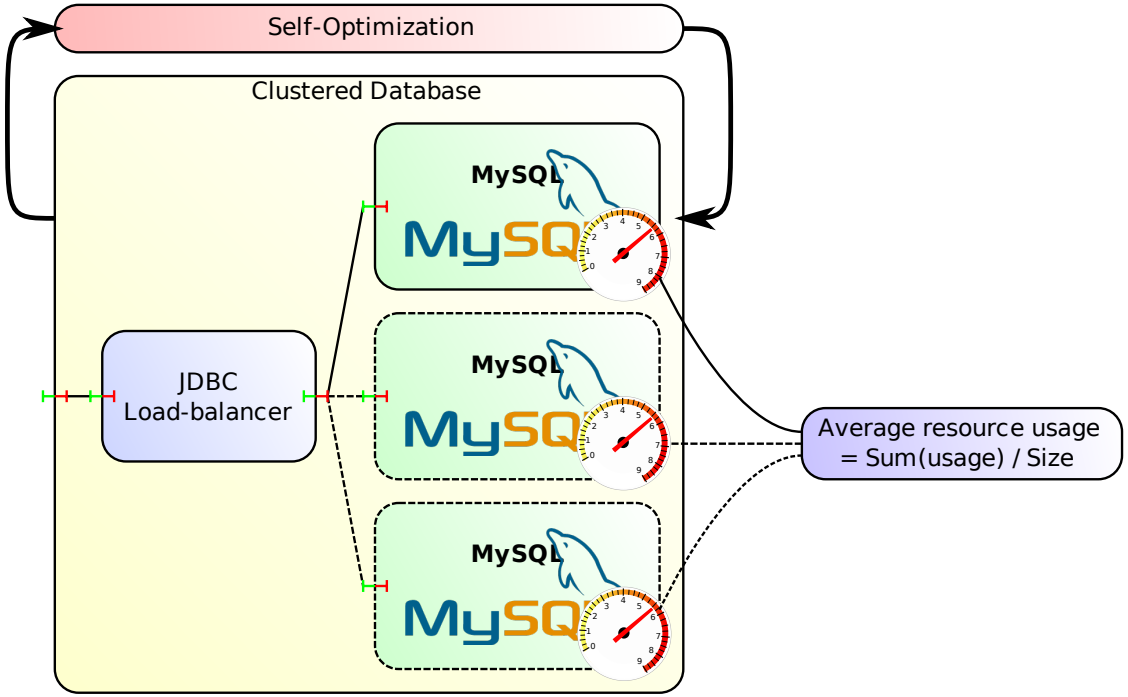
Design principles

- Dynamic architecture as a 1^{st} class citizen
- Dynamic computation model

Implementation

- Integration with FructOz
  - Fractal components
  - Architectural constructions
- LactOz library of navigation primitives

Clustered service monitoring
Example



Challenge

- Dynamic set of nodes to monitor
- Monitoring system adaptations

Dynamic architecture instrumentation
Related work

- Navigation and scripting DSL: Fractal FPath/FScript/ECA (Safran) [David, 2005]
  - Reactions only
- Active database: Triggers, Views, ECA rules
  - Not distributed & Database specific

Dynamic components as 1^{st} class citizen
Dynamic monitoring instrumentation


```
%% Dynamic monitoring architecture
fun {MonitoringPkg SComponents}
   %% Set of nodes hosting the components
   SHosts = {SMap SComponents CGetHost}
in
   functor $
   export Membrane
   define
      C = {CNew nil}

      %% Deploy the monitoring information aggregator
      Aggregator = {Deploy AggregatorPkg}
      {CAddSubComponent C Aggregator}

      %% Deploy a dynamic set of sensors on every host
      SSensors = {ClusteredDeploy SHosts SensorPkg}

      %% Bind all sensors to the aggregator
      SAggregators = {SSingleton Aggregator}
      {ApplyBindingScheme FullInterconnect SSensors SAggregator}

      Membrane = C
   end
end
```

Deployment on a dynamic set of hosts
Dynamic monitoring instrumentation


```
%% Deploy the given component (Package) on each host in the given set of hosts
%% Each component instance is made sub−component of the given Parent component
fun {ClusteredDeploy SHosts Package}
   %% Class for an implicit component set mapped over SHosts
   class ComponentSet from ImplicitSet
      ...
      %% Adding some hosts
      meth addHosts(SElements)
         {FSet.forAll SElements % for every new host
            proc {$ Host}
               % remotely deploy a component instance
               SubComp = {RemoteDeploy Host Package}
               % add the component to the implicit set
               Set,add(SubComp)
            end}
      end
      ...
   end
in
   %% Instantiate the implicit set
   {New ComponentSet init}
end
```

Conclusion

Summary of the problems

Problem

Self-Optimization of Autonomic Systems

- System provisioning optimization and coordination
- Distributed deployment of architectures
- Dynamic architecture instrumentation

Summary of the contributions

Jade self-optimization manager

- Simple threshold-based policy
- Architecture-based coordination

FructOz framework

- Higher Order ADL (Package)
- Synchronization of distributed architecture deployments

LactOz library

- Distributed dynamic computation model
- Dynamic architectures

Perspectives

Self-optimization policies

- Toward self-stabilizing control-loop parameters
- Modeling

FructOz & LactOz

- Transactional architecture manipulation
- (Self-)Optimization of dynamic computations
- Architecture checking (typing)
- Syntactic sugar (package, architectural entities, dynamic variables)

Main results

Code

Jade prototype:	Java code	23 kloc in 150 classes
	Fractal ADL	250 loc in 70 files
	Experimentations	20 nodes cluster
FructOz/LactOz:	Oz code	10 kloc in 90 files
	Experimentations	16 nodes cluster

Publications

- Bouchenak et al., Int. Conf. on Cluster Computing, 2006
- Taton et al., Int. Conf on Dist. Objects and Applications, 2007
- Bouchenak et al., Encyclopedia of Database Systems, 2008
- Taton et al., Int. Journal on Autonomic Computing, 2008
- Stefani and Taton, Technical Report, 2008

Appendix

Références I

E. Bruneton et al.

An Open Component Model and its Support in Java.
7th Int. Symposium on Component-Based Software Engineering, 2004.

P. Goldsack.

SmartFrog: Configuration, Ignition and Management of Distributed Applications.
Technical report, HP Research Labs, 2003.

K. Appleby et al.

Océano-SLA based management of a computing utility.
In *Proceedings of Integrated Network Management*, 2001.

G. Soundararajan and C. Amza.

Autonomic Provisioning of Backend Databases in Dynamic Content Web Servers.
Technical report, Dept. of Electrical and Computer Engineering, University of Toronto, 2005.

B. Urgaonkar and P. Shenoy.

Sharc: Managing CPU and Network Bandwidth in Shared Clusters.
IEEE Transactions on Parallel and Distributed Systems (TPDS), 2004.

M. Aron et al.

Cluster Reserves: a mechanism for resource management in cluster-based network servers.
In *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'00)*, 2000.

Références II

N. Arshad et al.

Deployment and Dynamic Reconfiguration Planning for Distributed Software Systems.
Software Quality Control Journal, 15(3):265–281, 2007.

P.-C. David.

Développement de composants Fractal adaptatifs: un langage dédié à l'aspects d'adaptation.
PhD thesis, Université de Nantes, France, 2005.