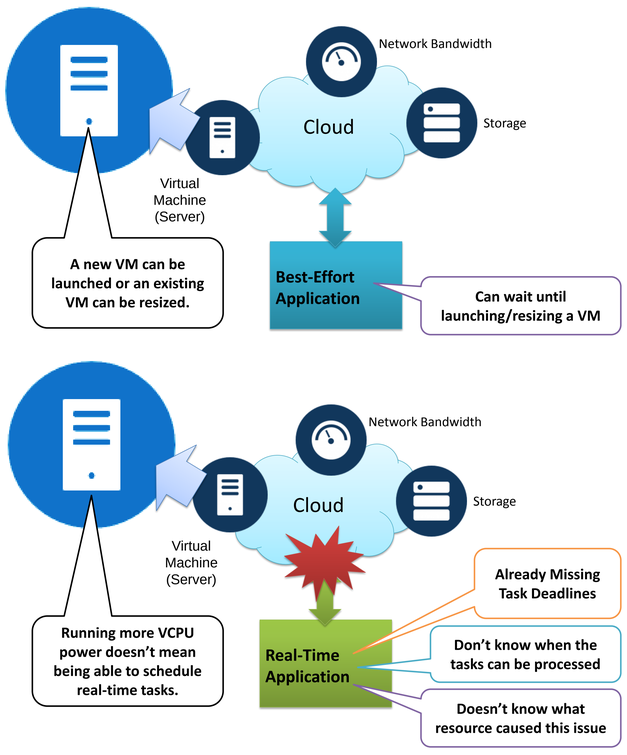
# MIRRA: Rule-Based Resource Management for Heterogeneous Real-Time Applications Running in Cloud Computing Infrastructures

Yong woon Ahn, Albert Mo Kim Cheng
Real-Time Systems Lab., Computer Science, University of Houston

## Virtual Resource Auto-Scaling for Real-Time Applications?




## Problems?

1. **SLA for cloud computing infrastructure does not** provide proper solutions to **schedule real-time tasks**.
2. Only **limited types of resource condition** can be determined by system administrators for the existing auto-scaling.
3. **Inevitable service-down time** unexpectedly.

## Requirements

- Identifying QoS downgrade issue for automatic performance monitoring
- Determining and monitoring whether they miss task deadlines or not
- Supporting an unknown number of heterogeneous real-time applications
- Monitoring application-specific QoS conditions

## Target Applications & Cloud Services

- **Applications transmitting periodic real-time tasks**
  - Real-time patient monitoring systems
  - Real-time Internet of Things device
  - Cyber Physical Systems
- **Public cloud infrastructures**
  - Amazon EC2, MS Azure, etc.
- **Private cloud infrastructures**
  - OpenStack, CloudStack, OpenNebula, etc.

## System Design

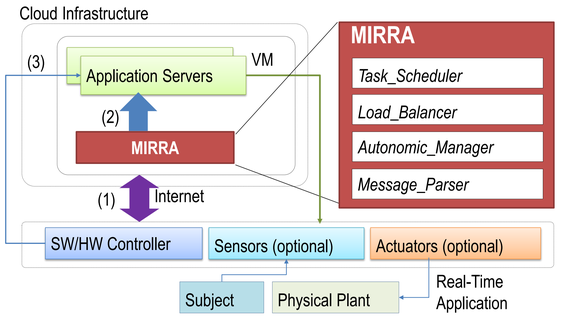



Figure 1. The system overview: (1) registering a new task, (2) scheduling the task by checking the current resource size, and (3) the application starts sending tasks to the assigned application server

## Real-Time Periodic Tasks

$$T_i = (S_i, C_i, O_i, M_i, B_i, d_i, ID_i),$$

The first task release time $S_i$,
The maximum computation time $C_i$,
The number of CPU cores $O_i$,
The maximum memory requirement $M_i$,
The maximum network bandwidth requirement $B_i$,
The relative deadline $d_i$
The task ID, $ID_i$.

## Service Interval

$$I = \{\max(d_i) \mid 0 < i < n\},$$

- For four steps of the autonomic manager.
- $n$ is the total number of registered real-time applications.

## System Design: Autonomic_Manager

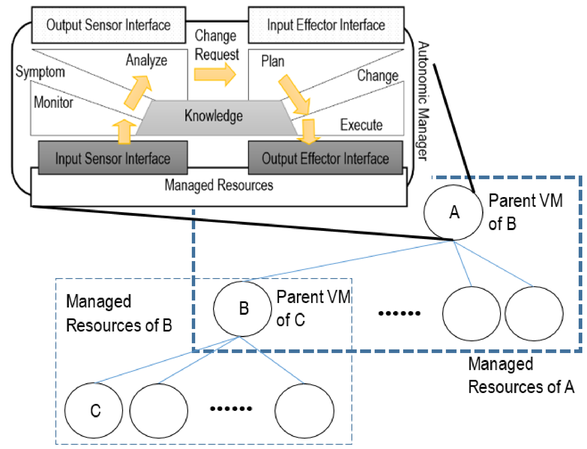



Figure 2. An autonomic computing architecture used to implement a class object of each VM stored and managed by MIRRA

## Reasoning and Reaction Rule

***A knowledge base contains:***

- ***Reasoning rules*** are used for identifying QoS issues
- ***Reaction rules*** are used for providing solutions for the issues
- Both rules are written in the ***RuleML*** format

```
rR_v[OID:"0"]:
{
    E  : "missing_deadline"
    VA : {X:uCPU_m}
    RE : {C:90%_uCPU }
    IF : {X⊥C}
    TH : "need_more_cores"
    L  : "cpu_check"
    SC : {App0}
    PR : "0"
}
```

**A reasoning rule example**

```
aR_v[OID:"2"]:
{
    E  : "need_more_cores"
    VA : {W:vm_t_1}
    RE : {Q:"available" }
    IF : {W⊥Q}
    DO : "launch_vm"
    L  : "launch for more
         cores"
    SC : {App0}
    PR : "0"
}
```

**A reaction rule example**

## Implementation & Evaluation

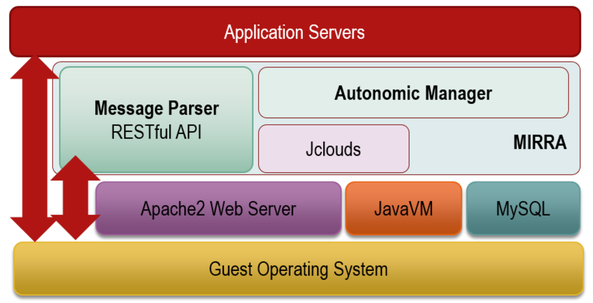



Figure 3. Software components running in a single VM

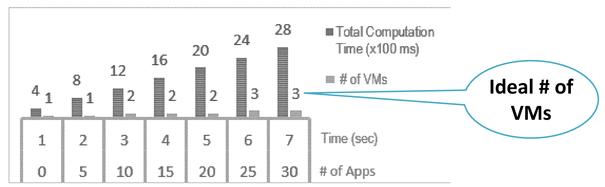



Figure 4. The total computation time to process real-time tasks successfully according to variation of the number of real-time applications

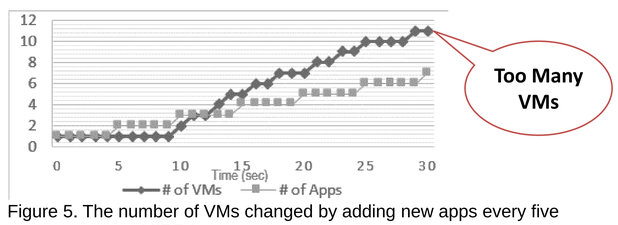



Figure 5. The number of VMs changed by adding new apps every five seconds without MIRRA

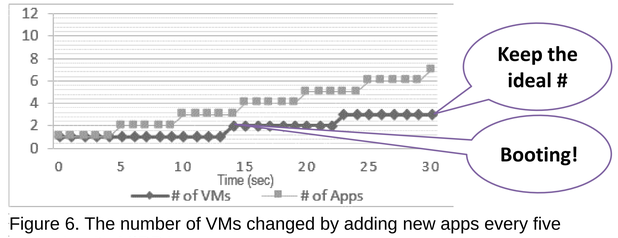



Figure 6. The number of VMs changed by adding new apps every five seconds with MIRRA

## Conclusion

- MIRRA is a new middleware solution adopting the autonomic computing and rule-based knowledge base to implement an auto-scaling mechanism for real-time applications.