Towards an Enactment Engine for Dynamically Reconfigurable and Scalable Choreographies

Thiago Furtado
Emilio Francesquini
Nelson Lago
Fabio Kon

Jun 2014
Introduction

The problem

The lack of elasticity and Quality of Services on the service composition layer

- Cloud environments provide their users with automatic elasticity at the virtual resource layer.
  - It is easy to allocate and deallocate resources, adding or removing computational nodes and migrating them
- We provide a middleware that makes use of this to offer composition owners a mechanism for self reconfiguration at the service layer.
  - QoS monitoring
  - Resource monitoring
Composition Model

- Services are abstract entities
- Instances are created on-demand
Quality of Services Properties

Distinct behavior of different services: Service Level Agreements designed for each service.

- **Response Time**: time taken between send a request and receive the response
- **Throughput**: Processed requests rate
- **Availability**: Percentage of the time a service is available
- **Cost**: The financial cost to run a service in a cloud environment

The middleware allows to configure each of the aspects above. Other aspects may be included
QoS-enabled Enactment Engine Architecture

- **MAPE** loop running in three phases
  - Monitoring
  - Analysis and Planning
  - Execution
Realization: Monitoring phase

To be capable of reconfiguring resources, we need to monitor both resources and services.

- Resource monitoring by means of Ganglia
  - Lightweight

- QoS metrics monitoring using interceptors in Tomcat
Realization: Analysis and Planning phases

- Measurements aggregation with complex event processing (CEP)
  - Correlation of events sent by Monitoring phase
  - Generation of complex events

- Policies and strategies
  - Scale Up: policies to migrate and **replicate**
  - Scale Down
  - Hosts availability checking
Realization: Rule example

when
    $ev : ResponseTimeEvent();
    HighCpuUser ($ev.ip)
    Number( $qtd : doubleValue ) from accumulate(
        $event : ResponseTimeEvent($ev.instance == instance, $ev.ip == ip),
        count($event) );
    Number( intValue > $qtd * 0.95 ) from accumulate(
        $sEvent : ResponseTimeEvent(value > 300, $ev.instance == instance, $ev.ip == ip),
        count($sEvent) );
then
    ResponseDispatcher.NotifyMeValue("ScaleUp", $ev.choreography, , $ev.service );
end
Realization: Execution phase (Update choreography)

- The enactment engine keeps choreography status metadata
- Update interface based on modified composition specification
- Identified modifications lead to specific Update Actions
  - **Horizontal Scale Up/Down**
  - Migration
  - Update service artifact
Experimental Setup

- Two services: one of them, instrumented to consume processing time
- Scale Up as response time gets higher than expected
- Avoid over/under allocated CPU capacity
- **Objective**: keep response time below 1000 ms and CPU usage between 60% and 95%

Platforms

- Enactment Engine Server: 2 cores 4GB - 2.53GHz
- Resource Manager Aggregator server: 1 virtual core 4GB - 2.25GHz
- Amazon AWS Nodes: 1 virtual core 1.7GB - 2.6GHz each
Experiment Results

- System reacts with high response time
- Unexpected: Virtual nodes of the same type not necessarily are identical
- Worse: configuration issues, apparently in network settings → Requests 10 times slower.
  - Partially solved by disabling reverse DNS lookups
- During modifications, unavailability time when reconfiguration is done
Towards an Enactment Engine for Dynamically Reconfigurable and Scalable Choreographies

Experimental Evaluation

Experiment

[Graph showing average response time, requests per second, and CPU usage over time for different replicas.]
Conclusion and Future Work

- We offer a extensible and flexible deployment framework
  - May be used as a testbed for experiments with service compositions

- This framework offers automated scalability management
  - Policies and strategies are configurable

- Node migration and elimination should take into account the actual performance of each node
  - Keep a runtime profile for each machine

- Use migration and replication together, comparison of results

- Minimize % of the time SLA is violated
Thank you.

Thiago Furtado
tfurtado@ime.usp.br