A QoS Assurance Framework for Distributed Infrastructures

André Lage Freitas, Nikos Parlavantzas, Jean-Louis Pazat

Université Européenne de Bretagne
INSA, INRIA, IRISA, UMR 6074
F-35708 Rennes, France

Myriads Team - INRIA–IRISA

3rd International Workshop on Monitoring, Adaptation and Beyond (MONA+)
Ayia Napa, Cyprus

December 1, 2010
Outline

1. Context
   - Service-Oriented Architecture (SOA)
   - The Problem
   - Challenges

2. QU4DS: Quality Assurance for Distributed Services
   - Architecture
   - Use Cases
   - Implementation
   - Preliminary Evaluation

3. Conclusions
1. Context
   - Service-Oriented Architecture (SOA)
   - The Problem
   - Challenges

2. QU4DS: Quality Assurance for Distributed Services
   - Architecture
   - Use Cases
   - Implementation
   - Preliminary Evaluation

3. Conclusions
Service-Oriented Architecture (SOA) [5]

- Service abstraction
  - Distributed business applications
  - Re-usability
  - Interoperability
  - Loose-coupling

- Service-Based Applications (SBA)
  - Composition of services
  - Service-Level Agreements (SLA)
    - Functional
    - Non-functional
Quality of Service (QoS)
- Promotes business
- Differentiates service providers
- Influences directly on contract establishments

Examples:
- Request response time
- Request throughput
- Service availability
The Problem

- To ensure the agreed QoS
- Prevent SLA violations
  - Avoid penalties
  - Reduce costs
  - Improve service reputation
The Problem: Scope

- **Atomic** service on large-scale distributed infrastructures

![Diagram of Service-Based Application]

- **Service-Based Application**

  - **M**
  - **W**
  - **A**
  - **B**
  - **C**
  - **D**
  - **E**
  - **W**
  - **W**
  - **W**
  - **W**
- Provide **QoS assurance** for **atomic** service on large-scale distributed infrastructures
Goal

- Provide **QoS assurance** for **atomic** service on large-scale distributed infrastructures
● Provide **QoS assurance** for **atomic** service on large-scale distributed infrastructures
• Provide **QoS assurance** for **atomic** service on large-scale distributed infrastructures
- Provide **QoS assurance** for **atomic** service on large-scale distributed infrastructures
Raised Issues

- What can the underlying infrastructure provide?
- How to match QoS requirements to service instantiation and resource configurations?
- How to deal with the dynamic environment?
What can the underlying infrastructure provide?

- Trade-off
  - Simplicity
  - Control

- A Uniform Infrastructure Usage

- SAGA [3] easy + accounting

- Separation of two distinct managements
  - Job
  - Resource
What can the underlying infrastructure provide?

- Trade-off
  - Simplicity
  - Control

- A Uniform Infrastructure Usage

- SAGA [3] easy + accounting

- Separation of two distinct managements
  - Job
  - Resource

- Simple and high-level
  - Job
    - create(jobDescription)
    - run()
    - cancel()
    - checkpoint()
    - suspend()
    - resume()
    - migrate()
    - registerCallback()

- Accounting
  - Pricing model
How to match QoS requirements to service instantiation and resource configurations?

- QoS ↔ service instance and resource configurations
  - E.g.: response time ↔ instantiation requirements + number of resources

- It is not trivial

- High-level infrastructures interfaces are not enough
How to match QoS requirements to service instantiation and resource configurations?

- QoS ↔ service instance and resource configurations
  - E.g.: response time ↔ instantiation requirements + number of resources

- It is not trivial

- High-level infrastructures interfaces are not enough

Representation of such a translation

- Application profiling
- Analytical models
- Implementation details

- The more knowledge, the more accurate
How to deal with the dynamic environment?

- Unpredictability of service demand
  - New customers
  - Provision changes

- Infrastructure dynamism
  - Availability
  - Requirements fluctuations
How to deal with the dynamic environment?

- Unpredictability of service demand
  - New customers
  - Provision changes

- Infrastructure dynamism
  - Availability
  - Requirements fluctuations

**Self-Adaptation**
- Adaptation at runtime
- Autonomic control loop [4]
- Dynaco [1]

![Diagram of self-adaptation process]

- Analysis
- Monitor
- Planning
- Execution

- Policies
- Strategy
- Guides plan

André Lage Freitas (INRIA - Myriads Team)  Quality Assurance for Distributed Services  December 2010  16 / 32
Uniform Interface
Infrastructure
The Pieces Together

QoS Translation

Uniform Interface

Infrastructure
Outline

1 Context
   • Service-Oriented Architecture (SOA)
   • The Problem
   • Challenges

2 QU4DS: Quality Assurance for Distributed Services
   • Architecture
   • Use Cases
   • Implementation
   • Preliminary Evaluation

3 Conclusions
QU4DS is a framework for ensuring QoS for distributed services
QU4DS: Quality Assurance for Distributed Services

- QU4DS is a framework for ensuring QoS for distributed services
- Translate QoS parameters to service instance and resource configurations in a bi-directional way
- Automatically deploy the service on appropriate resources
- Ensure the agreed QoS by reacting to underlying infrastructures changes while keeping compliant to the QoS objectives
QU4DS – Architecture

Service Provider

QoS Translator

SLA Negotiator

Instantiator

QoS Assurance Controller

Analysis

Monitor

Planning

Execution

policies

strategy

guides plan

QU4DS

Infrastructure API

Infrastructure

Instance 1

r₁

r₂

r₃

r₄

r₅

Instance 2
QU4DS – Use Case 2: QoS Assurance

Service Provider

QoS Translator

SLA Negotiator

Instantiator

QoS Assurance Controller

Analysis

Monitor

Planning

Execution

policies

strategy

QoS model

Service Customer

Interface

Infrastructure API

Infrastructure

r₁

r₂

r₃

r₄

r₅

Instance 1

Instance 2

migration

11

12

13

14

15

16

17

19

18
Implementation Overview

- Prototype implemented in Java
- Support for Master/Worker applications
- Management of service tasks
  - Workers wrapped as jobs
  - Master can focus on its main concern
Implementation Overview

- Prototype implemented in Java
- Support for Master/Worker applications
- Management of service tasks
  - Workers wrapped as jobs
  - Master can focus on its main concern
- Infrastructure: simple, high-level API and based on XOSAGA
  - Backend: XtreemOS [2]
  - Customized metrics
QoS Translator: application profiling

- QoS ↔ # of workers
  - E.g.: 1 min response time ↔ 4 workers
Implementation Details

- **QoS Translator: application profiling**
  - QoS $\leftrightarrow$ # of workers
  - E.g.: 1 min response time $\leftrightarrow$ 4 workers

- **Self-Adaptation: QoS Assurance Controller**
  - Implements a simple control loop
  - **Monitor**
    - Job: state, elapsed time, CPU usage, number of threads, command
    - QoS: response time
  - **Decider**
    - Event-condition-action
  - **Planning**
    - Sequence of XOSAGA methods
  - **Executor**
Case Study: The flac2ogg Service

- An audio encoder
  - Encodes Flac to Ogg
  - Master/Worker service

- QoS
  - Response time
  - Translation
    - Degree of parallelization
    - $t = 23 \text{ sec/MB} \leftrightarrow 12 \text{ workers}$

- Adaptation strategy
  - Single Replacement for Late Jobs (SRLJ)

<table>
<thead>
<tr>
<th>Policies</th>
<th>Conditions</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j$: jobETimeThreshold</td>
<td>if (jobETime &gt; $j$) AND (requestETime &lt; $rt$)</td>
<td>1) create a job to replace the late job</td>
</tr>
<tr>
<td>$rt$: respTimeThreshold</td>
<td></td>
<td>2) cancel the late job</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3) submit the job replacement</td>
</tr>
</tbody>
</table>
Preliminary Evaluation

- Emulated environment (virtual machine)
  - 2.4 GHz CPU
  - 1.5 GB memory
  - XtreemOS core and resource

- Two faulty scenarios
  - **NA-F**: Non-Adaptable
  - **A-F**: Adaptable
  - QoS translation
    - response time 500 secs ↔ 12 workers
  - 30 requests
Preliminary Evaluation

- Emulated environment (virtual machine)
  - 2.4 GHz CPU
  - 1.5 GB memory
  - XtreemOS core and resource
- Two faulty scenarios
  - **NA-F**: Non-Adaptable
  - **A-F**: Adaptable
  - QoS translation
    - response time 500 secs ↔ 12 workers
  - 30 requests

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Violated requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>NA-F</td>
<td>12 (40%)</td>
</tr>
<tr>
<td>A-F</td>
<td>6 (20%)</td>
</tr>
</tbody>
</table>

Table: QU4DS reduced to half the number of SLA violations.
1 Context
   - Service-Oriented Architecture (SOA)
   - The Problem
   - Challenges

2 QU4DS: Quality Assurance for Distributed Services
   - Architecture
   - Use Cases
   - Implementation
   - Preliminary Evaluation

3 Conclusions
Conclusions

- **QU4DS**: a framework for quality assurance of distributed services
  - Prevents SLA violations
  - Re-negotiates agreements
  - Eases the development of distributed and QoS-aware services

- **Self-Adaptation**
  - Autonomic control loop
  - Single Replacement for Late Jobs (SRLJ)

- **Prototype**
  - XtreemOS
  - Early results are promising
Current and Future Directions

On-going work

- Uniform infrastructure interface under XOSAGA and Grid’5000
- Improvements on integrating monitoring mechanisms and the infrastructure
- Manage distinct contracts in parallel

Challenges for future work

- QoS Translation accuracy
  - Service provider knowledge
- Support beyond Master/Worker applications
  - E.g.: workflow management
- Negotiate resource usage with the infrastructure
Thank you!

References


