A QoS-Aware Middleware for Fault Tolerant Web Services

Zibin Zheng and Michael R. Lyu

Department of Computer Science & Engineering
The Chinese University of Hong Kong
Hong Kong, China

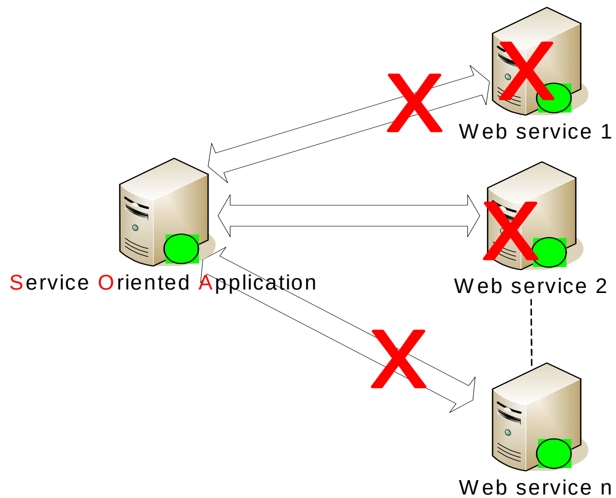
ISSRE 2008, Seattle, USA, 11-14 November, 2008

Outlines

1. Introduction
2. A QoS-Aware Middleware
3. Fault Tolerance Strategies
4. Dynamic Strategy Selection Algorithms
5. Experiments
6. Conclusion and Future Work

1. Introduction

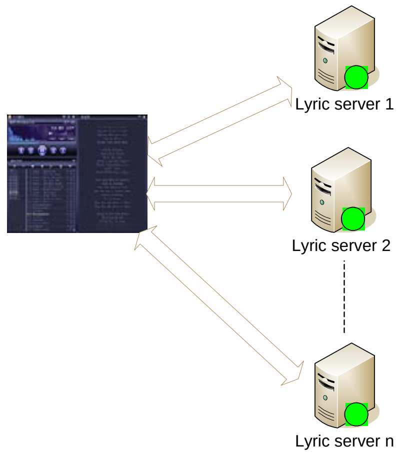
- Web services are becoming popular.
- Reliability of the service-oriented applications becomes difficult to be guaranteed.
  - Remote Web services may contain faults.
  - Remote Web services may become unavailable.
  - The Internet environment is unpredictable.



1. Introduction

- Traditional software reliability engineering
  - Fault Tolerance is a major approach for building highly reliable system.
  - Expensive.
- Service reliability engineering
  - Abundant Web service candidates with identical/similar interface.
  - Less expensive & less time-consuming.
- The Internet environment is highly dynamic
  - Network condition changes.
  - Software/hardware updates of the Web services.
  - Server workload changes.

1. Introduction



For a service user:

- Design time:
  1. Which Web service is the best to choose?
  2. What are the available fault tolerance strategies?
  3. Which fault tolerance strategy is optimal?
- Run time:
  3. How to automatically determine the optimal fault tolerance strategy in a highly dynamic environment?

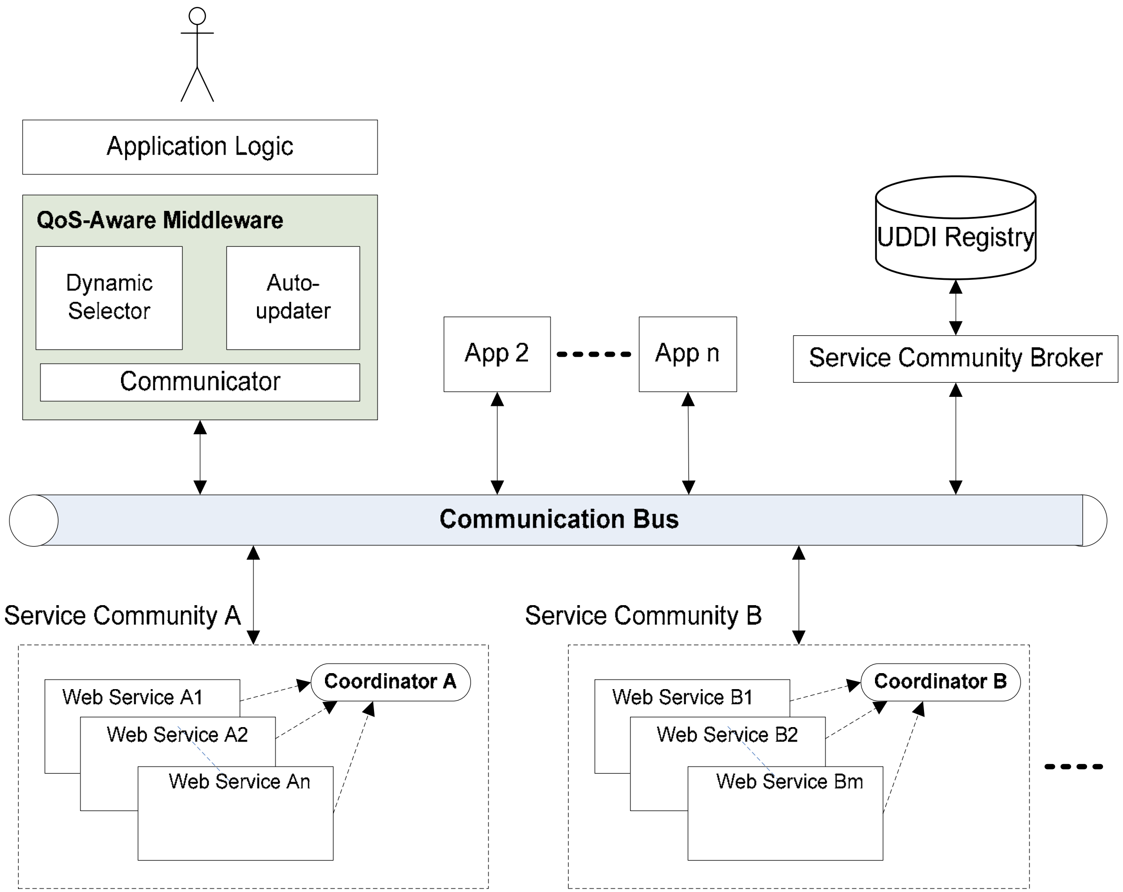
1. Introduction

- A QoS-Aware Middleware for Fault Tolerant (FT) Web Services.
  - A user-collaborated QoS model
    - *YouTube*: sharing videos.
    - *Wikipedia*: sharing knowledge.
    - Sharing QoS information of target Web services.
  - Record QoS information of target Web services and exchange it with other service users
  - Determine the optimal fault tolerance strategy dynamically at runtime based on the QoS information

2. QoS-Aware Middleware

- The need for overall QoS information (different locations and access time) of target Web services:
  - Service users
    - Web service selection and ranking.
    - Optimal fault tolerance strategy selection.
  - Service providers
    - Performance of their own Web service from different users.
    - Providing better services.
- The overall QoS information is difficult to obtain
  - Time-consuming
  - Expensive

2. QoS-Aware Middleware



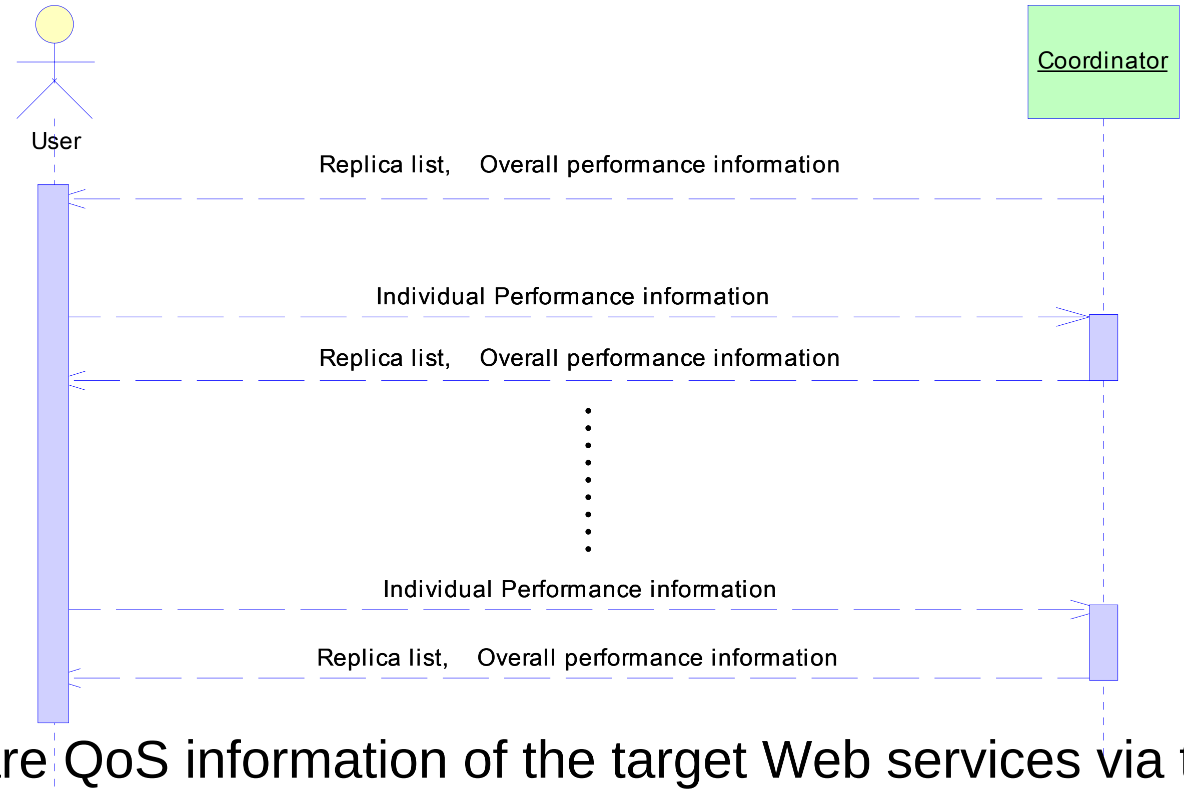
1. Coordinator address.
2. Replica list and QoS.
3. Optimal FT strategy.
4. Record QoS data.
5. Exchange QoS data.
6. Adjust for the optimal FT strategy.

User-collaborated QoS-Aware Middleware

2. QoS-Aware Middleware

- How to obtain functional identical Web services?
  - Machine learning techniques for automatic identification.
  - Service Communities: define a common interface so that the Web services provided by different organizations have the same functionality, although with different levels of non-functional quality of service (QoS).

2. QoS-Aware Middleware



- Users share QoS information of the target Web services via the coordinator of the service community.
- WS-DREAM: Web Service Distributed REliability Assessment Mechanism.
- Middleware: users can close the data exchange functionality.
- BitTorrent: users can close the upload.

3. Fault Tolerance Strategies

f: failure rate *t*: access time

- Retry

$$f = f_1^m; \qquad t = \sum_{i=1}^{m} t_i (f_1)^{i-1}$$

- Recovery Block

$$f = \prod_{i=1}^{m} f_i; \qquad t = \sum_{i=1}^{m} t_i \prod_{k=1}^{i-1} f_k$$

3. Fault Tolerance Strategies

- N-Version Programming (NVP)

$$f = \sum_{i=v/2+1}^{v} F(i); \qquad t = \max(\{t_i\}_{i=1}^{v})$$

- Active

$$f = \prod_{i=1}^{u} f_i; t = \begin{cases} \min(T_c) : |T_c| > 0 \\ \max(T) : |T_c| = 0 \end{cases}$$

3. Fault Tolerance Strategies

- Dynamic sequential strategy (Retry+RB)

$$f = \prod_{i=1}^{n} f_i^{m_i}; t = \sum_{i=1}^{n} ((\sum_{j=1}^{m_i} t_i f_i^{j-1}) \prod_{k=1}^{i-1} f_k^{m_i})$$

- Dynamic parallel strategy (NVP+Active)

$middle(v, T_c)$: u replicas in parallel, first v for voting.

$$f = \sum_{i=v/2+1}^{v} F(i); t = \begin{cases} middle(v, T_c) : |T_c| \geq v \\ \max(T) : |T_c| < v \end{cases}$$

4. Selection Algorithm

User requirements:

t_{max}: the largest RTT that the application can afford.

f_{max}: the largest failure-rate that the application can tolerate.

r_{max}: the largest resource consumption constraint.

$mode$: the mode can be set by the service users to be *sequential*, *parallel*, or *auto*.

4. Selection Algorithm

The QoS model:

- t_{avg} : the average RTT of the target replica.
- t_{std} : the standard deviation of RTT of the target replica.
- fl : the logic failure-rate of the target replica.
- fn : the network failure-rate of the target replica.

4. Selection Algorithm

- The users may not be willing to store a lot of historical data.
- Without historical data, it is difficult to make QoS predictions.

Solution: Store the distribution

- Dividing the time t_{max} into k timeslots.
- k+2 counters for k timeslots, fl and fn.
- $p_i = \frac{c_i}{\sum_{i=1}^{k+2} c_i}$ for calculating the probability of a certain RTT belongs to a certain category.

4. Selection Algorithm

RTT Prediction:

Problem 1 *Given:*

- $\{ws_i\}_{i=1}^{v}$: a set of target replicas for prediction.
- $\{p_{i,j}\}_{j=1}^{k+2}$: for replica i ($1 \leq i \leq v$), the probability of an RTT belonging to different categories.
- $\{t_i\}_{i=1}^{k}$: the RTT value of the time slot i, which can be calculated by $t_i = (t_{max} \times i)/k - t_{max}/(2 \times k)$.
- $T_v = \{rtt_j\}_{j=1}^{v}$: a set of RTT of the v replicas, where the probability of rtt_j belonging to the time slot k is provided by $p_{j,k}$.

Find out:

- $\mathrm{E}(\min(T_v))$: the average response time by invoking all the v replicas in parallel for many times, where function $\min(T_v)$ stands for the minimal RTT value of all the $\{rtt_j\}_{j=1}^{v}$.

4. Selection Algorithm

RTT Prediction:

$$\mathrm{E}(\min(T_v)) = \sum_{i=1}^{k} (\mathrm{P}(\min(T_v) == t_i) \times t_i)$$

$$\mathrm{P}(\min(T_v) == t_i) = \mathrm{P}(\min(T_v) \leq t_i) - \mathrm{P}(\min(T_v) \leq t_{i-1})$$

$$\mathrm{P}(\min(T_v) \leq t_i) = \mathrm{P}(rtt_n \leq t_i) + \mathrm{P}(rtt_n > t_i) \times \mathrm{P}(\min(T_{v-1}) \leq t_i)$$

$$\mathrm{P}(rtt_i \leq t_j) = \sum_{k=1}^{j} p_{i,k}$$

min(Tv): Active strategy.
max(Tv): NVP.
middle(Tv, x): v parallel replicas and employs the first x response for voting.

4. Selection Algorithm

- Sequential or parallel strategy determination:

$$p_i = \frac{t_i}{t_{max}} + \frac{f_i}{f_{max}} + \frac{r_i}{r_{max}}$$

- Dynamic sequential strategy determination:

Degradation factor $d = \frac{1}{m} \times (\frac{t_{i+1} - t_i}{t_{max}} + \frac{f_{i+1} - f_i}{f_{max}})$

- Dynamic parallel strategy determination:
  - RTT prediction algorithm
  - Combination numbers: $C_n^v = \frac{n!}{v! \times (n-v)!}$

5. Experiments

- The experimental system is implemented by JDK6.0, Eclipse3.3, Axis2.0, and Tomcat6.0.
- Developed six Web services following an identical interface to simulate replicas in a same service community.
- The six Web services and the community coordinator are deployed on seven PCs.
  - Pentium(R) 4 CPU 2.8 GHz, 1G RAM;
  - 100Mbits/sec Ethernet card;
  - Windows XP operating system.

5. Experiments

Table 1. Service Users and Requirements

Users	t_{max}	f_{max}	r_{max}	Focus
User 1	1000	0.1	50	RTT
User 2	2000	0.01	20	RTT, Fail
User 3	4000	0.03	2	RTT, Fail, Res
User 4	10000	0.02	1	Res
User 5	15000	0.005	3	Fail, Res
User 6	20000	0.0001	80	Fail

Table 2. Parameters of Experiments

	Parameters	Setting
1	Number of replicas	6
2	Network fault probability	0.01
3	Logic fault probability	0.0025
4	Permanent fault probability	0.05
5	Number of time slots	20
6	Performance degradation threshold (a)	2
7	Replica number of NVP	5
8	Parallel replica number of $Active$	6
9	Dynamic degree	20

Table 3. Experimental Results of User 1

U	Strategies	All	RTT	Fail	Res	Perf
1	Retry	50000	420	2853	1	1.011
	RB	50000	420	2808	1	1.002
	NVP	50000	839	2	5	0.939
	Active	50000	251	110	6	0.393
	Dynamic	50000	266	298	2.34	0.372

- The new *Dynamic* approach gets the best overall performance.
- Similar to the *Active* strategy.
- With good RTT performance for User 1.

5. Experiments

Table 1. Service Users and Requirements

Users	t_{max}	f_{max}	r_{max}	Focus
User 1	1000	0.1	50	RTT
User 2	2000	0.01	20	RTT, Fail
User 3	4000	0.03	2	RTT, Fail, Res
User 4	10000	0.02	1	Res
User 5	15000	0.005	3	Fail, Res
User 6	20000	0.0001	80	Fail

Table 4. Experimental Results of User 2

U	Strategies	All	RTT	Fail	Res	Perf
2	Retry	50000	471	285	1	5.985
	RB	50000	469	283	1	5.944
	NVP	50000	855	0	5	0.677
	Active	50000	253	126	6	2.946
	Dynamic	50000	395	3	4.03	0.459

Table 5. Experimental Results of User 3

U	Strategies	All	RTT	Fail	Res	Perf
3	Retry	50000	458	155	1	0.717
	RB	50000	457	149	1	0.713
	NVP	50000	845	1	5	2.712
	Active	50000	248	138	6	3.154
	Dynamic	50000	456	141	1	0.708

Table 7. Experimental Results of User 5

U	Strategies	All	RTT	Fail	Res	Perf
5	Retry	50000	454	115	1	0.823
	RB	50000	450	121	1	0.847
	NVP	50000	779	0	5	1.718
	Active	50000	249	125	6	2.516
	Dynamic	50000	489	60	1.46	0.759

Table 6. Experimental Results of User 4

U	Strategies	All	RTT	Fail	Res	Perf
4	Retry	50000	498	145	1	1.194
	RB	50000	493	131	1	1.180
	NVP	50000	868	1	5	5.087
	Active	50000	251	119	6	6.144
	Dynamic	50000	494	109	1	1.158

Table 8. Experimental Results of User 6

U	Strategies	All	RTT	Fail	Res	Perf
6	Retry	50000	470	146	1	29.236
	RB	50000	468	119	1	23.835
	NVP	50000	839	1	5	0.304
	Active	50000	249	132	6	26.487
	Dynamic	50000	473	1	3.56	0.2682

5. Experiments

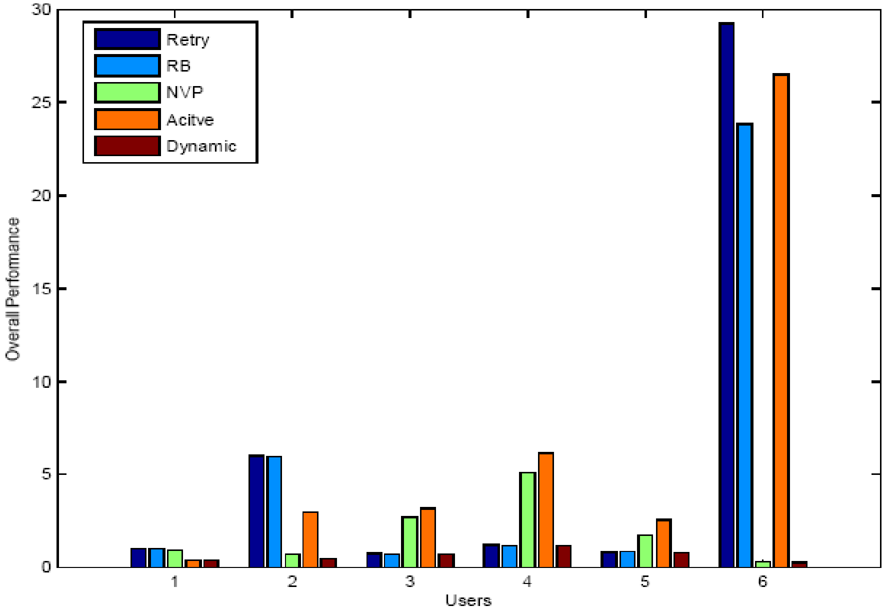


Figure 3. Overall Performance of Strategies

$$p_i = \frac{t_i}{t_{max}} + \frac{f_i}{f_{max}} + \frac{r_i}{r_{max}}$$

1. Traditional static fault tolerance strategies do not get good results consistently.
2. The proposed dynamic strategy obtains the best overall performance for all the six users in the experiments.

6. Conclusion and Future Work

- Conclusion
  - An innovative QoS-aware middleware approach was proposed for reliable Web services
    - Dynamic fault tolerance replication strategies.
    - Dynamic replication strategy selection algorithm.
  - Encouraging experimental results were obtained.

- Future work
  - Investigating more QoS properties.
  - Evaluation of stateful Web services.

A QoS-Aware Middleware for Fault Tolerant Web Services

Zibin Zheng and Michael R. Lyu

The Chinese University of Hong Kong
Hong Kong, China

Questions?