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.
- Middleware: users can close the data exchange functionality.
- BitTorrent: users can close the upload.
3. Fault Tolerance Strategies

\[ f: \text{failure rate} \quad t: \text{access time} \]

- **Retry**

\[ f = f_1^m; \quad t = \sum_{i=1}^{m} t_i (f_1)^{i-1} \]

- **Recovery Block**

\[ f = \prod_{i=1}^{m} f_i; \quad t = \sum_{i=1}^{m} t_i \prod_{k=1}^{i-1} f_k \]
3. Fault Tolerance Strategies

• N-Version Programming (NVP)

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

• Active

\[ f = \prod_{i=1}^{u} f_i; \quad t = \begin{cases} \min(T_c) : |T_c| > 0 \\ \max(T) : |T_c| = 0 \end{cases} \]
3. Fault Tolerance Strategies

- Dynamic sequential strategy (Retry+RB)
  \[ f = \prod_{i=1}^{n} f_{i}^{m_{i}}; \quad t = \sum_{i=1}^{n} \left( \sum_{j=1}^{m_{i}} t_{i} f_{i}^{j-1} \right) \prod_{k=1}^{i-1} f_{k}^{m_{i}} \]

- Dynamic parallel strategy (NVP+Active)
  \[ middle(v, T_{c}) : u \text{ replicas in parallel, first } v \text{ for voting.} \]
  \[ f = \sum_{i=v/2+1}^{v} F(i); \quad t = \begin{cases} middle(v, T_{c}) : |T_{c}| \geq v \\ \max(T) : |T_{c}| < v \end{cases} \]
4. Selection Algorithm

User requirements:

\( t_{\text{max}} \): the largest RTT that the application can afford.

\( f_{\text{max}} \): the largest failure-rate that the application can tolerate.

\( r_{\text{max}} \): the largest resource consumption constraint.

\( \text{mode} \): 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:*

- \( \mathbb{E}(\min(T_v)) \): 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:

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

\[
P(\min(T_v) = t_i) = P(\min(T_v) \leq t_i) - P(\min(T_v) \leq t_{i-1})
\]

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

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

\(\min(T_v)\): Active strategy.

\(\max(T_v)\): NVP.

\(\text{middle}(T_v, x)\): \(v\) parallel replicas and employs the first \(x\) response for voting.
4. Selection Algorithm

• Sequential or parallel strategy determination:

\[ p_i = \frac{t_i}{t_{\text{max}}} + \frac{f_i}{f_{\text{max}}} + \frac{r_i}{r_{\text{max}}} \]

• Dynamic sequential strategy determination:

Degradation factor \[ d = \frac{1}{m} \times \left( \frac{t_{i+1}-t_i}{t_{\text{max}}} + \frac{f_{i+1}-f_i}{f_{\text{max}}} \right) \]

• 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

The new Dynamic approach gets the best overall performance.

Similar to the Active strategy.

With good RTT performance for User 1.

<table>
<thead>
<tr>
<th>Users</th>
<th>$t_{max}$</th>
<th>$f_{max}$</th>
<th>$r_{max}$</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>1000</td>
<td>0.1</td>
<td>50</td>
<td>RTT</td>
</tr>
<tr>
<td>User 2</td>
<td>2000</td>
<td>0.01</td>
<td>20</td>
<td>RTT, Fail</td>
</tr>
<tr>
<td>User 3</td>
<td>4000</td>
<td>0.03</td>
<td>2</td>
<td>RTT, Fail, Res</td>
</tr>
<tr>
<td>User 4</td>
<td>10000</td>
<td>0.02</td>
<td>1</td>
<td>Res</td>
</tr>
<tr>
<td>User 5</td>
<td>15000</td>
<td>0.005</td>
<td>3</td>
<td>Fail, Res</td>
</tr>
<tr>
<td>User 6</td>
<td>20000</td>
<td>0.0001</td>
<td>80</td>
<td>Fail</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Number of replicas</td>
<td>6</td>
</tr>
<tr>
<td>2 Network fault probability</td>
<td>0.01</td>
</tr>
<tr>
<td>3 Logic fault probability</td>
<td>0.0025</td>
</tr>
<tr>
<td>4 Permanent fault probability</td>
<td>0.05</td>
</tr>
<tr>
<td>5 Number of time slots</td>
<td>20</td>
</tr>
<tr>
<td>6 Performance degradation threshold</td>
<td>2</td>
</tr>
<tr>
<td>7 Replica number of NVP</td>
<td>5</td>
</tr>
<tr>
<td>8 Parallel replica number of Active</td>
<td>6</td>
</tr>
<tr>
<td>9 Dynamic degree</td>
<td>20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Strategies</th>
<th>U</th>
<th>All</th>
<th>RTT</th>
<th>Fail</th>
<th>Res</th>
<th>Perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retry</td>
<td>1</td>
<td>50000</td>
<td>420</td>
<td>2853</td>
<td>1</td>
<td>1.011</td>
</tr>
<tr>
<td>RB</td>
<td>1</td>
<td>50000</td>
<td>420</td>
<td>2808</td>
<td>1</td>
<td>1.002</td>
</tr>
<tr>
<td>NVP</td>
<td>1</td>
<td>50000</td>
<td>839</td>
<td>2</td>
<td>5</td>
<td>0.939</td>
</tr>
<tr>
<td>Active</td>
<td>1</td>
<td>50000</td>
<td>251</td>
<td>110</td>
<td>6</td>
<td>0.393</td>
</tr>
<tr>
<td>Dynamic</td>
<td>1</td>
<td>50000</td>
<td>266</td>
<td>298</td>
<td>2.34</td>
<td>0.372</td>
</tr>
</tbody>
</table>
5. Experiments

Table 1. Service Users and Requirements

<table>
<thead>
<tr>
<th>Users</th>
<th>$t_{max}$</th>
<th>$f_{max}$</th>
<th>$r_{max}$</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>1000</td>
<td>0.1</td>
<td>50</td>
<td>RTT</td>
</tr>
<tr>
<td>User 2</td>
<td>2000</td>
<td>0.01</td>
<td>20</td>
<td>RTT, Fail</td>
</tr>
<tr>
<td>User 3</td>
<td>4000</td>
<td>0.03</td>
<td>2</td>
<td>RTT, Fail, Res</td>
</tr>
<tr>
<td>User 4</td>
<td>10000</td>
<td>0.02</td>
<td>1</td>
<td>Res</td>
</tr>
<tr>
<td>User 5</td>
<td>15000</td>
<td>0.005</td>
<td>3</td>
<td>Fail, Res</td>
</tr>
<tr>
<td>User 6</td>
<td>20000</td>
<td>0.0001</td>
<td>80</td>
<td>Fail</td>
</tr>
</tbody>
</table>

Table 4. Experimental Results of User 2

<table>
<thead>
<tr>
<th>U</th>
<th>Strategies</th>
<th>All</th>
<th>RTT</th>
<th>Fail</th>
<th>Res</th>
<th>Perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Retry</td>
<td>50000</td>
<td>471</td>
<td>285</td>
<td>1</td>
<td>5.985</td>
</tr>
<tr>
<td></td>
<td>RB</td>
<td>50000</td>
<td>469</td>
<td>283</td>
<td>1</td>
<td>5.944</td>
</tr>
<tr>
<td></td>
<td>NVP</td>
<td>50000</td>
<td>855</td>
<td>0</td>
<td>5</td>
<td>0.677</td>
</tr>
<tr>
<td></td>
<td>Active</td>
<td>50000</td>
<td>253</td>
<td>126</td>
<td>6</td>
<td>2.946</td>
</tr>
<tr>
<td></td>
<td>Dynamic</td>
<td>50000</td>
<td>395</td>
<td>3</td>
<td>4.03</td>
<td>0.459</td>
</tr>
</tbody>
</table>

Table 5. Experimental Results of User 3

<table>
<thead>
<tr>
<th>U</th>
<th>Strategies</th>
<th>All</th>
<th>RTT</th>
<th>Fail</th>
<th>Res</th>
<th>Perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Retry</td>
<td>50000</td>
<td>458</td>
<td>155</td>
<td>1</td>
<td>0.717</td>
</tr>
<tr>
<td></td>
<td>RB</td>
<td>50000</td>
<td>457</td>
<td>149</td>
<td>1</td>
<td>0.713</td>
</tr>
<tr>
<td></td>
<td>NVP</td>
<td>50000</td>
<td>845</td>
<td>1</td>
<td>5</td>
<td>2.712</td>
</tr>
<tr>
<td></td>
<td>Active</td>
<td>50000</td>
<td>248</td>
<td>138</td>
<td>6</td>
<td>3.154</td>
</tr>
<tr>
<td></td>
<td>Dynamic</td>
<td>50000</td>
<td>456</td>
<td>141</td>
<td>1</td>
<td>0.708</td>
</tr>
</tbody>
</table>

Table 7. Experimental Results of User 5

<table>
<thead>
<tr>
<th>U</th>
<th>Strategies</th>
<th>All</th>
<th>RTT</th>
<th>Fail</th>
<th>Res</th>
<th>Perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Retry</td>
<td>50000</td>
<td>454</td>
<td>115</td>
<td>1</td>
<td>0.823</td>
</tr>
<tr>
<td></td>
<td>RB</td>
<td>50000</td>
<td>450</td>
<td>121</td>
<td>1</td>
<td>0.847</td>
</tr>
<tr>
<td></td>
<td>NVP</td>
<td>50000</td>
<td>779</td>
<td>0</td>
<td>5</td>
<td>1.718</td>
</tr>
<tr>
<td></td>
<td>Active</td>
<td>50000</td>
<td>249</td>
<td>125</td>
<td>6</td>
<td>2.516</td>
</tr>
<tr>
<td></td>
<td>Dynamic</td>
<td>50000</td>
<td>489</td>
<td>60</td>
<td>1.46</td>
<td>0.759</td>
</tr>
</tbody>
</table>

Table 6. Experimental Results of User 4

<table>
<thead>
<tr>
<th>U</th>
<th>Strategies</th>
<th>All</th>
<th>RTT</th>
<th>Fail</th>
<th>Res</th>
<th>Perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Retry</td>
<td>50000</td>
<td>498</td>
<td>145</td>
<td>1</td>
<td>1.194</td>
</tr>
<tr>
<td></td>
<td>RB</td>
<td>50000</td>
<td>493</td>
<td>131</td>
<td>1</td>
<td>1.180</td>
</tr>
<tr>
<td></td>
<td>NVP</td>
<td>50000</td>
<td>868</td>
<td>1</td>
<td>5</td>
<td>5.087</td>
</tr>
<tr>
<td></td>
<td>Active</td>
<td>50000</td>
<td>251</td>
<td>119</td>
<td>6</td>
<td>6.144</td>
</tr>
<tr>
<td></td>
<td>Dynamic</td>
<td>50000</td>
<td>494</td>
<td>109</td>
<td>1</td>
<td>1.158</td>
</tr>
</tbody>
</table>

Table 8. Experimental Results of User 6

<table>
<thead>
<tr>
<th>U</th>
<th>Strategies</th>
<th>All</th>
<th>RTT</th>
<th>Fail</th>
<th>Res</th>
<th>Perf</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Retry</td>
<td>50000</td>
<td>470</td>
<td>146</td>
<td>1</td>
<td>29.236</td>
</tr>
<tr>
<td></td>
<td>RB</td>
<td>50000</td>
<td>468</td>
<td>119</td>
<td>1</td>
<td>23.835</td>
</tr>
<tr>
<td></td>
<td>NVP</td>
<td>50000</td>
<td>839</td>
<td>1</td>
<td>5</td>
<td>0.304</td>
</tr>
<tr>
<td></td>
<td>Active</td>
<td>50000</td>
<td>249</td>
<td>132</td>
<td>6</td>
<td>26.487</td>
</tr>
<tr>
<td></td>
<td>Dynamic</td>
<td>50000</td>
<td>473</td>
<td>1</td>
<td>3.56</td>
<td>0.2682</td>
</tr>
</tbody>
</table>
5. Experiments

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.

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

Figure 3. Overall Performance of Strategies
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?

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