Transaction Support for Log-Based Middleware Server Recovery

Presented by XiaoFei Zhao
• Introduction.
• *Middleware server process* (MSP) architecture.
• State recovery based on logging.
• State recovery based on results logging.
• Caveats about database management systems (DBMSs).
• Related work.
• Conclusion.
• Middleware servers.
• High availability and exactly-once semantics.
• Recovery after failure.
• Middleware server process & concurrency.
• Transactional methods.
• Results logging.
• Low overhead and little or no change to back-end infrastructure.
Sessions & clients
One request per session
Session variables vs. shared variables
Transactional methods & transactional requests
Committed transactions vs. aborted transactions
Local transactions vs. distributed transactions
Strict two phase locking

Fig. 1. Transaction-extended MSP architecture.
State recovery based on logging

Log all non-deterministic events
Log shared variables, do not log session variables
Replay log to recover
Session variables: re-execute; shared variables: read log
Take checkpoints

<table>
<thead>
<tr>
<th>execution scenario</th>
<th>log records</th>
</tr>
</thead>
<tbody>
<tr>
<td>execute recoverable method 1</td>
<td>request for method 1 with parameters</td>
</tr>
<tr>
<td>read session variable sev1</td>
<td>value of sev1 being read</td>
</tr>
<tr>
<td>read shared variable shv1</td>
<td></td>
</tr>
<tr>
<td>write sev1</td>
<td></td>
</tr>
<tr>
<td>write shv1</td>
<td></td>
</tr>
<tr>
<td>execute recoverable method 2</td>
<td>new value of sev2 being written</td>
</tr>
<tr>
<td>read session variable sev2</td>
<td></td>
</tr>
<tr>
<td>write sev2</td>
<td>request for method 2 with parameters</td>
</tr>
<tr>
<td>write shared variable shv1</td>
<td></td>
</tr>
<tr>
<td>new value of shv1 being written</td>
<td></td>
</tr>
</tbody>
</table>
Log the results of execution
Read most recent values of both shared vars and session vars to recover
Save the whole in-memory state

**TABLE II**

A transactional method execution and results logging.

<table>
<thead>
<tr>
<th>execution scenario</th>
<th>logging</th>
</tr>
</thead>
<tbody>
<tr>
<td>execute transactional method 1 read session variable sev1 read shared variable shv1 open a DBMS connection read/write database write sev1 write shv1 return retv</td>
<td>request log record for method 1 committing log record: retv, new values of sev1 and shv1 flush log buffer send commit decision to TranMan wait for transaction outcome result-status log record: committed or aborted</td>
</tr>
<tr>
<td>automatically close connection</td>
<td></td>
</tr>
</tbody>
</table>
Caveats about database management systems (DBMSs).

- Cannot re-execute transactional methods.
- Flush log buffer before committing.
- Let the transaction manager remember committed transactions’ status until being explicitly told to discard such status.
- Use distributed log flush with dependency vector.
• Fault tolerance via replication.
• E-transactions.
• Phoenix project: message logging.
• Transactional Web methods.
Conclusion

- Transaction support for log-based recovery of middleware servers.
- Results logging can recover both in-memory business state and persistent business state while incurring modest overhead and requiring almost no change to existing transaction systems.
- Results logging and existing transaction system recovery facilities take care of system failures.
Thank you!

XiaoFei Zhao
Discussion

• If the middleware crashes but all transactions were recorded in a persistent storage medium which survived the crash, then can we recover the system to its state just before the last recorded transaction?

• What are the strengths and weaknesses of the replication-based and log-based recoveries respectively?