Reverb: Middleware for Distributed Application Forensics

Patrick M. Widener
College of Computing
Georgia Institute of Technology
pmw@cc.gatech.edu

www.cercs.gatech.edu
Outline / Contributions

- Context and motivation
- Description of Reverb
  - Differential, customizable, access-controlled auditing for distributed middleware
- Application example
- Experimental results
  - Small performance overhead
  - Preserves application scalability
- Concluding remarks
Example problem domain – scientific application

**Cluster Computer**

Application Services

<table>
<thead>
<tr>
<th>Transform</th>
<th>Differentiate</th>
<th>Specialize</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedicated Science Network</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>End Users and Displays</td>
<td></td>
<td></td>
</tr>
<tr>
<td>High End Users and Displays</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Instrumented Testbeds/Facilities</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data reduction/fusion
Information creation/manipulation

**Services**
- dynamically deployed, cooperating components
- heterogeneous platforms, devices, users

**Example problem domain** – scientific application
Context / motivation

- Application trends
  - Large scale, component-based, dynamically configurable / extensible

- Configuration changes can raise issues
  - System integrity, performance effects, responsibility for outages

- Audit tools for configuration changes help
  - “Paper trail”, on-line or off-line forensic analysis

- Perform audit actions *differentially, dynamically*
  - Differential change control: who can initiate which changes?
  - Differential auditing: which changes are audited, and who sees the audit trail?
  - Change constraints at run-time, without taking applications off-line
Reverb: Dynamic, differential control

- Reverb provides mechanism to
  - Track dynamic configuration actions
  - Impose controls on permissible actions (which / who)
  - Control access to audit trail

- Dedicated event channel for configuration events (RChannel)
  - Access controlled
  - Differential customization of configuration events

- Implemented in publish/subscribe middleware
  - “ECho” provides customizable event channel abstraction (EChannel)

- Monitored events: channel creation/destruction, subscription, channel customization
Reverb in action

• Sensors, visualization from large science application

Installation of image filters (greyscale, edge detection, etc)
Reverb auditing

- **Publisher**
  - **RChannel**
    - **CUSTOMIZATION**
    - **SUBSCRIPTION**
    - **CREATION**
  - **downsample**
    - **color**
    - **greyscale**
  - **EChannel**
    - **Subscriber**

- **RChannel subscriber**

**Reverb events**
- Submitted to RChannel at publisher
- No user intervention (auto-submit by middleware)
- Distributed to all RChannel subscribers
Differential auditing / change control

- Should any / all users have access to customizations to the RChannel?
- Policy-driven access to RChannel, customizations
  - Per-principal, per-customization basis - differential
- Reverb provides protected access
  - ECho protected mode - capabilities required
  - Dennis & Van Horn style - reference + rights
  - Cryptographic protection against forgery / replay
  - Trusted policy module (Overwatch) to issue / sign capabilities
- Configuration policy statements (XML) at startup, during execution
  - Policy statements can dictate differential actions
  - Overwatch creates differential code, RChannel references
Specifying customizations

- Coarse-grain: by configuration type
  - CREATION, SUBSCRIPTION, CUSTOMIZATION, etc.
- Fine-grain: based on application spec
- How to specify? How to execute?
  - Dynamically compiled filter functions
    - Safe(r) subset of C
    - Execution at source
    - Satisfies large % of needs
  - DLL / shared objects
    - More complex filtering
    - Stateful

- Configuration event structures published in API
Reverb policy statements

```xml
<userPolicy>
  <name> Bob </name>
  <Reverb-restrictions>
    <auditDisallowed>
      CREATION
    </auditDisallowed>
  </Reverb-restrictions>
</userPolicy>

<customization takesParams="false">
  <name> greyImage </name>
  <code>...</code>
  <Reverb-actions>
    <disallowUser>
      Bob
    </disallowUser>
  </Reverb-actions>
</customization>
```

Authentication Database

Overwatch

Customization metadata

Capability for code / RChannel
Customized, protected
RChannels

- More efficient event propagation
- Applications define exactly what information goes to what principals (least privilege)
- Subdivide audit processing
  - Monolithic audit de-multiplexer unwieldy, complex
  - Instead, small audit components, each with specialized task
- Dynamic policy reactions at Overwatch
  - Disable customizations for suspect users
  - Disable suspect customization code
Reverb protection overhead

- Protection mechanisms profiled against unmodified middleware
- Overheads
  - Channel create, subscribe, filter uninstall are small (3-5%)
  - Customization larger (~8%), but more XML, communication, crypto
- Most overheads outside data path - cost is amortized
Reverb Scalability

(5 Reverb clients, action script)

![Graph showing the relationship between the number of ECho subscribers and time for total send time, system CPU time, and user CPU time.](image)
Reverb Scalability - Multiple Client Customizations
(20/80)

![Graph showing the relationship between the number of Reverb clients and time.
Legend: total send time, system CPU time, user CPU time.
The x-axis represents the number of Reverb clients, ranging from 10 to 50.
The y-axis represents time (s), ranging from 0 to 2.
There are three curves: one for total send time in blue, one for system CPU time in magenta, and one for user CPU time in yellow.
The curves show an increasing trend as the number of clients increases.]
Conclusion

- This talk has described Reverb
  - Middleware mechanism to support auditing and forensics for large distributed applications

- Customizable, protected, efficient dissemination of configuration information
  - Customizable - Subscribers choose which configuration events they wish to see
  - Protected - only principals authorized by application policy can access RChannel
  - Efficient - tolerable overheads, scalability as Reverb and non-Reverb subscribers increase

- Dynamic, differential auditing, change control
- Part of larger work on data protection in high-performance, pervasive applications
Questions?