WAP5
Black-box Performance Debugging for Wide-Area Distributed Systems

Patrick Reynolds
reynolds@cs.duke.edu

With:
Janet Wiener    Marcos Aguilera
Jeffrey Mogul   Amin Vahdat

http://www.hpl.hp.com/research/project5/
Motivation

- Discover structure and performance problems in large, wide-area systems
- Infer paths through nodes
  - One path per client request
  - Discover timing at each step
- Focus attention on nodes that are problematic
  - First step in performance debugging
Coral example

- **Causal path:** a sequence of related messages and processing, annotated with timing/delays

- Second-level hit (4 messages)
- Second-level miss (6 messages)
- Also: DHT lookups
Goals

• **Find bugs in wide-area applications**
  – Performance bugs: too much or too little time at any point
  – Structure bugs: incorrect ordering or placement of processing or communication

• **Expose causal paths**
  – Structure discovery
  – Measure latency for processing and communication
  – Unexpected structure or timing
    • Indicates possible bugs

• **Black-box approach**
  – Do not require source code access
  – Allow heterogeneity
Three target audiences

• **Primary programmer**
  – Debugging or optimizing his/her own system

• **Secondary programmer**
  – Inheriting a project or joining a programming team
  – **Discovery**: learning how the system behaves

• **Operator**
  – Monitoring a running system for unexpected behavior
  – Performing regression tests after a change
Contributions

- **New causality analysis algorithm**
- **Full tool chain**
  - Trace capture library
  - Causal path analysis
  - Visualization
- **Results with two PlanetLab CDNs**
  - Coral and CoDeeN
Outline

- Introduction
- **Naming**
- Trace capture
- Reconciliation
- Causality analysis
  - Message linking algorithm
- Results with CoDeeN & Coral
Naming

- Message is single read/write system call
  - May be many TCP or UDP packets
- Node can be process or host
- Endpoint can be socket path or <IP address, port>

Host = foo.cs.duke.edu

Client \(\rightarrow\) Web proxy (pid=2297)

Web proxy \(\rightarrow\) Web server (pid=2297)

Web server \(\rightarrow\) DHT node (pid=2312)

DHT node \(\rightarrow\) /tmp/corald...

(pid=2297)

(pid=2312)
Naming

- **Node names are causal names**
  - Message into a process/host can cause messages out

- **Endpoint names guide aggregation**
  - Calls to `foo:8080` are different from calls to `foo:53`
  - Client hosts and ports can be ignored

```
Client 1025 8080 Web proxy pid=2297 1207 80 Web server
     |                 /
     |                /
     |               /tmp/corald...
DHT 1207 pid=2312
     |                 /
     |                /
     |               /tmp/corald...
DHT node
```

Host = foo.cs.duke.edu
Outline

• Introduction
• Naming
• **Trace capture**
• Reconciliation
• Causality analysis
  – Message linking algorithm
• Results with CoDeeN & Coral
Trace capture

- Capture events using host/net sniffing or library interposition
  - All three choices: no modifications to applications
  - On PlanetLab: sniffing on host only, limited flexibility
- We capture events using library interposition
  - Captures all calls that create, modify, or use a socket
Outline

• Introduction
• Naming
• Trace capture
• **Reconciliation**
• Causality analysis
  – Message linking algorithm
• Results with CoDeeN & Coral
Reconciliation: Convert socket calls to logical messages

- Assign endpoint names to each call

```plaintext
client pid=5040
bind(fd=6, addr={15.1.2.3:33250})
connect(fd=6, addr={16.5.6.7:80})
send(fd=6, len=10, time=0.592)
recv(fd=6, len=12, time=2.033)

server pid=8712
bind(fd=4, addr={16.5.6.7:80})
accept(lfd=4, addr={15.1.2.3:33250}) = 5
recv(fd=5, len=10, time=0.852)
send(fd=5, len=12, time=1.705)
```

```plaintext
client/5040 server/8712 0.592 0.852
server/8712 client/5040 1.705 2.033
```
Reconciliation: Convert socket calls to logical messages

- Combine *send* and *recv* events for each message
  - Detect dropped or reordered UDP packets
  - Detect differing message (buffer) boundaries

| client pid=5040 | bind(fd=6, addr={15.1.2.3:33250})
|                 | connect(fd=6, addr={16.5.6.7:80})
|                 | send(fd=6, len=10, time=0.592)
|                 | recv(fd=6, len=12, time=2.033) |
| server pid=8712 | bind(fd=4, addr={16.5.6.7:80})
|                 | accept(lfd=4, addr={15.1.2.3:33250}) = 5
|                 | recv(fd=5, len=10, time=0.852)
|                 | send(fd=5, len=12, time=1.705) |

client/5040 server/8712 0.592 0.852
server/8712 client/5040 1.705 2.033
Reconciliation:
Convert socket calls to logical messages

• Assign node (process) names to each message

client
bind(fd=6, addr={15.1.2.3:33250})
connect(fd=6, addr={16.5.6.7:80})
send(fd=6, len=10, time=0.592)
recv(fd=6, len=12, time=2.033)

server
bind(fd=4, addr={16.5.6.7:80})
accept(lfd=4, addr={15.1.2.3:33250}) = 5
recv(fd=5, len=10, time=0.852)
send(fd=5, len=12, time=1.705)

cliente/5040 server/8712 0.592 0.852
server/8712 cliente/5040 1.705 2.033
Outline

• Introduction
• Naming
• Trace capture
• Reconciliation
• Causality analysis
  – Message linking algorithm
• Results with CoDeeN & Coral
Causal path analysis

• Which call to B caused outgoing calls?
  – Could be spontaneous action
  – May be ambiguous
    • Make good guesses
    • Use statistics over whole trace
    • Try multiple possibilities

• Build paths by combining calls
Message linking algorithm

Message traces

Estimate average causal delays

Score possible parents for each message

Link-probability trees

Build and aggregate paths

Causal-path patterns
Estimate average causal delay

- Look at all messages into B, plus all B→C messages
  - Take smallest delay before each B→C message
  - Trace-specific upper limit
- \( D_{B\rightarrow C} = \text{average of these delays} \)
  - Might underestimate D
- Scaling factor \( \lambda_{B\rightarrow C} = 1/D_{B\rightarrow C} \)
- Create exponential distribution
  - \( f(t) = \lambda e^{-\lambda t} \)
Find and weight possible parent messages

- Use $f(t)$ to find weight of link from each parent

![Graph showing weight function $f(t)$ over delay $t$.](image)

- $f(t_4-t_3) = 0.368$
- $f(t_4-t_2) = 0.135$
- $f(t_4-t_1) = 0.050$
- $f(4) = 0.018$
Find and weight possible parent messages

- Normalize so sum of weights to each child = 1
- Possible-parent trees
  - Spontaneous action has small probability, not shown
  - Links to B→D are slightly less likely
Build causality trees

- Invert to get possible-child trees
Build causality trees

• Build trees from individual links
  – Use probability to decide whether or not to keep child
  – Some links are “try-both” and generate 2 trees
• Tree probability is product of link probabilities
  \[ p = 0.8 \times 0.9 \times (1-0.2) \times (1-0.1) \times (1-0.48) \approx 0.270 \]
Build causality trees

• **Aggregate trees with identical structure**
  – Combine client names and ports for better aggregation

• **Total probabilities for each pattern ➔ ranking**
  – Expected number of instances
  – Highlights paths that appear **many times with high confidence**
Outline

• Introduction
• Naming
• Trace capture
• Reconciliation
• Causality analysis
  – Message linking algorithm
• Results with CoDeeN & Coral
Results: Timeline vs. call tree

- Coral miss path with DNS lookup

```
<table>
<thead>
<tr>
<th></th>
<th>PU</th>
<th>PU</th>
<th>web6</th>
<th>web6</th>
</tr>
</thead>
<tbody>
<tr>
<td>cl</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dns3</td>
<td>2.2</td>
<td></td>
<td>51.7</td>
<td></td>
</tr>
<tr>
<td>Coral processing</td>
<td>64.7</td>
<td></td>
<td>10.9</td>
<td></td>
</tr>
<tr>
<td>Coral processing</td>
<td></td>
<td>48.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client</td>
<td></td>
<td>64.7 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Server (PU)</td>
<td></td>
<td>178.2 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dns2</td>
<td>2.2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>web6</td>
<td></td>
<td>51.7 ms</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Origin server Response
Results: Two CoDeeN miss paths

- Different mean delays at proxies
  - 0.20 to 4.86 ms in different proxies
- Different delays at origin web servers
- All clients aggregated together
Results: Coral DHT lookup

- Three-level DHT lookups

3 calls in parallel
Conclusions

• **WAP5** exposes structure and timing of wide-area applications
  – Particularly PlanetLab applications

• **Successful analysis of CoDeeN and Coral traces**
  – We found paths that match authors’ descriptions of systems
  – We characterized delays at each step and found outliers

http://www.hpl.hp.com/research/project5/
Extra slides
Future work

- **Phased behavior**
  - Time-varying patterns
  - Time-varying delays

- **Better aggregation**
  - Coalesce similar path patterns
    - Paths through DHTs

- **Better visualization**
Related work

• **Trace-based analysis tools**
  – Our LAN-based work in SOSP 2003
  – Magpie: detailed picture per machine by using OS-level instrumentation (Event Tracing for Windows)
  – Pinpoint: instrument middleware
  – Others instrument applications

• **Inference-based performance analysis**
  – SLIC uses statistical induction to correlate low-level metrics with SLO violations

• **Interposition tools**
  – Trickle, ModelNet
Node and network latency

- **Node latency** = $t_3 - t_2$
  - Not affected by clock offset
  - All timestamps are local to B
- **Network latency** = $t_2 - t_1$; $t_4 - t_3$
  - Correct for clock offset \([\text{Paxson}98]\)
  - $\text{RTT} = (t_4 - t_3) + (t_2 - t_1)$
  - $\text{Skew} = (t_2 - t_1) - \text{RTT}/2$
LibSockCap interposition library

- **Low overhead**
- **Easy to deploy in CoDeeN and Coral**
  - Use existing framework to push out new software
  - Restart process to begin/end trace
- **Advantages**
  - Logical message semantics
  - Per process, not per machine
  - Capture UNIX, TCP, and UDP sockets
- **Disadvantages**
  - Timestamps combine OS and network latency
  - No control packets or fragments