Performance Debugging for Distributed Systems of Black Boxes

Marcos K. Aguilera
Jeffrey C. Mogul
Janet L. Wiener
HP Labs
Patrick Reynolds, Duke
Athicha Muthitacharoen, MIT
Example multi-tier system
Motivation

- Complex distributed systems are built from black box components
- These systems may have performance problems
  - High or erratic latency
- Locating the causes of these problems is hard
  - We can’t always examine or modify system components
- We need tools to infer where bottlenecks are
  - Choose which black boxes to open
Contributions of our work

• Tools to highlight which black boxes to open
  • Require only passive information, such as packet traces
  • Infer where most of time is spent from traces
• Person can then use more invasive tools to instrument those boxes
• Reduce time and cost to debug complex systems
• Improve quality of delivered systems
Example causal path
Goals of our tools

• Find high-impact causal paths through a distributed system
  
  **Causal path:** series of nodes that sent/received messages
  - Each message is caused by receipt of previous message
  - Some causal paths occur many times
  
  **High impact:**
  - Occurs frequently
  - Contributes significantly to overall latency

• Without modifications or semantic knowledge
• Report per-node latencies on causal paths
Overview of our approach

- Obtain traces of messages between components
  - Ethernet packets, middleware messages, etc.
  - Collect traces as non-invasively as possible
- Analyze traces using our algorithms
  - Nesting: faster, more accurate, limited to RPC-style systems
  - Convolution: works for all message-based systems
- Visualize results and highlight high-impact paths
- Require very little information:
  [timestamp, source, destination, call/return, call-id]
Outline

• Problem statement & goals
• Overview of our approach
• Algorithms
  - The nesting algorithm
  - The convolution algorithm
• Experimental results
• Visualization GUI
• Related work
• Conclusions
The nesting algorithm

- Uses traces with call-return semantics
- Infers causality from “nesting” relationships
  - Suppose A calls B and B calls C before returning to A
  - Then the B↔C call is “nested” in the A↔B call
Nesting: an example causal path

Consider this system of 4 nodes
Looking for internal delays at each node

![Diagram showing causal paths between nodes A, B, C, and D with call and return arrows over time.](image)
Steps of the nesting algorithm

1. Pair call and return messages
   - (A⇒B, B⇒A), (B⇒D, D⇒B), (B⇒C, C⇒B)

2. Find and score all nesting relationships
   - B⇒C nested in A⇒B
   - B⇒D also nested in A⇒B

3. Pick best parents
   - Here: unambiguous

4. Reconstruct call paths
   - A⇒B⇒[C ; D]

O(m) run time
m = number of messages
Inferring nesting

- Parallel calls are tricky
  - Local info not enough
  - Use aggregate info
  - Histograms keep track of possible latencies
    - One histogram per node triplet \(<A, B, C>\)
  - Two passes over trace
    - Build histograms
    - Assign nesting
  - Other heuristics in paper
Outline

• Problem statement & goals
• Overview of our approach
• **Algorithms**
  - The nesting algorithm
  - The convolution algorithm
• Experimental results
• Visualization GUI
• Related work
• Conclusions
The convolution algorithm

• “Time signal” of messages for each `<source node, destination node>`
  - A sent message to B at times 1,2,5,6,7

\[ S_1(t) = A \rightarrow B \text{ messages} \]
The convolution algorithm

• Look for time-shifted similarities
  - Compute convolution $X(t) = S_2(t) \otimes S_1(-t)$
  - Use Fast Fourier Transforms

Peaks in $X(t)$ suggest causality between $A \rightarrow B$ and $B \rightarrow C$

Time shift of a peak indicates delay
Convolution details

- Time complexity: $O(em + eV\log V)$
  - $m = \text{messages}$
  - $e = \text{output edges}$
  - $V = \text{number of time steps in trace}$
- Need to choose time step size
  - Must be shorter than delays of interest
  - Too coarse: poor accuracy
  - Too fine: long running time
- Robust to noise in trace
Algorithm comparison

• Nesting
  - Looks at individual paths and then aggregates
  - Finds rare paths
  - Requires *call/return* style communication
  - Fast enough for real-time analysis

• Convolution
  - Applicable to a broader class of systems
  - Slower: more work with less information
  - May need to try different time steps to get good results
  - Reasonable for off-line analysis

• More details in paper
Outline

- Problem statement & goals
- Overview of our approach
- Algorithms
  - Experimental results
    - Maketrace: a trace generator
    - Maketrace web server simulation
    - Pet Store EJB traces
    - Execution costs
- Visualization GUI
- Related work
- Conclusions
Maketrace

- Synthetic trace generator
- Needed for testing
  - Validate output for known input
  - Check corner cases
- Uses set of causal path templates
  - All call and return messages, with latencies
  - Delays are $x \pm y$ seconds, Gaussian normal distribution
- Recipe to combine paths
  - Parallelism, start/stop times for each path
  - Duration of trace
Desired results for one trace

- **Causal paths**
  - How often
  - How much time spent

- **Nodes**
  - Host/component name
  - Time spent in node and all of the nodes it calls

- **Edges**
  - Time parent waits before calling child
Experimental results for same trace

Correct

False positives < 20%

21% of next frequent path

8% of total paths
Results: Petstore

- Sample EJB application
- J2EE middleware for Java
  - Instrumentation from Stanford’s PinPoint project
- Delay added in mylist.jsp
## Results: running time

<table>
<thead>
<tr>
<th>Trace</th>
<th>Length (messages)</th>
<th>Duration (sec)</th>
<th>Memory (MB)</th>
<th>CPU time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nesting</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-tier (short)</td>
<td>20,164</td>
<td>50</td>
<td>1.5</td>
<td>0.23</td>
</tr>
<tr>
<td>Multi-tier</td>
<td>202,520</td>
<td>500</td>
<td>13.8</td>
<td>2.27</td>
</tr>
<tr>
<td>Multi-tier (long)</td>
<td>2,026,658</td>
<td>5,000</td>
<td>136.8</td>
<td>23.97</td>
</tr>
<tr>
<td>PetStore</td>
<td>234,036</td>
<td>2,000</td>
<td>18.4</td>
<td>2.92</td>
</tr>
<tr>
<td><strong>Convolution (20 ms time step)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PetStore</td>
<td>234,036</td>
<td>2,000</td>
<td>25.0</td>
<td>6,301.00</td>
</tr>
</tbody>
</table>

More details and results in paper
Accuracy vs. parallelism

- Increased parallelism degrades accuracy slightly
- Parallelism is number of paths active at same time
Other results for nesting algorithm

- **Clock skew**
  - Little effect on accuracy with skew • delays of interest

- **Drop rate**
  - Little effect on accuracy with drop rates • 5%

- **Delay variance**
  - Robust to • 30% variance

- **Noise in the trace**
  - Only matters if same nodes send noise
  - Little effect on accuracy with • 15% noise
Visualization GUI

- **Goal:** highlight dominant paths
- **Paths sorted**
  - By frequency
  - By total time
- **Red highlights**
  - High-cost nodes
- **Timeline**
  - Nested calls
  - Dominant subcalls
- **Time plots**
  - Node time
  - Call delay
Related work

• Systems that trace end-to-end causality via modified middleware using modified JVM or J2EE layers
  - Magpie (Microsoft Research), aimed at performance debugging
  - Pinpoint (Stanford/Berkeley), aimed at locating faults
  - Products such as AppAssure, PerformaSure, OptiBench

• Systems that make inferences from traces
  - Intrusion detection (Zhang & Paxson, LBL) uses traces + statistics to find compromised systems
Future work

• Automate trace gathering and conversion
• Sliding-window versions of algorithms
  - Find phased behavior
  - Reduce memory usage of nesting algorithm
  - Improve speed of convolution algorithm
• Validate usefulness on more complicated systems
• What are limits of our approach?
Conclusions

• Looking for bottlenecks in black box systems
• Finding causal paths is enough to find bottlenecks
• Algorithms to find paths in traces really work
  - We find correct latency distributions
  - Two very different algorithms get similar results
  - Passively collected traces have sufficient information
• For more information
• Contact us if you have multi-hop message traces!