Performance Debugging for Distributed Systems of Black Boxes

Yinghua Wu, Haiyong Xie

Outline

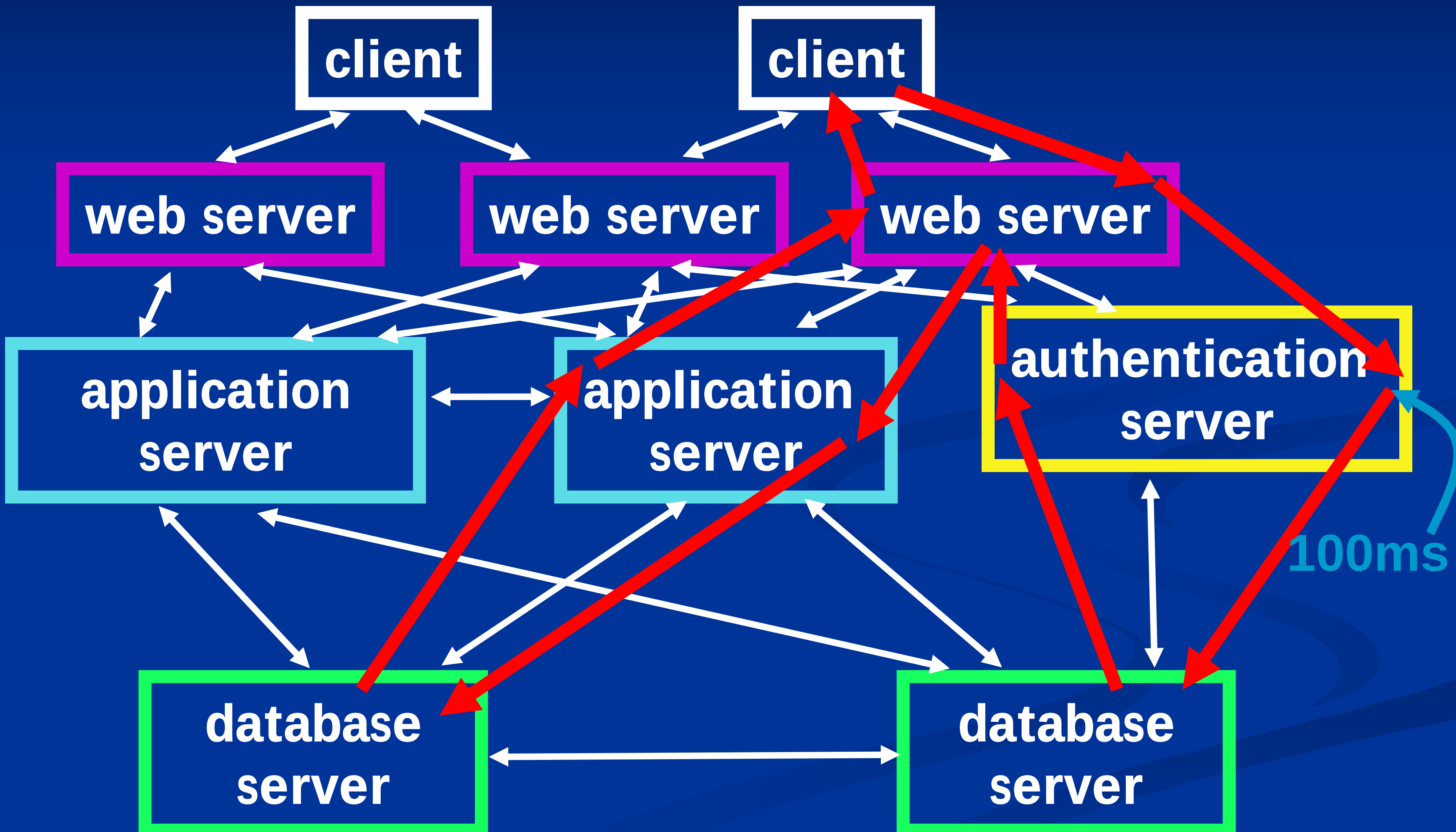
- Problem statement & goals
- Overview of our approach
- Algorithms
  - The nesting algorithm (RPC)
  - The convolution algorithm (RPC or free-form)
- Experimental results
- Visualization GUI
- Related work
- Conclusions

Motivation

- Complex distributed systems
  - Built from black box components
  - Heavy communications traffic
  - Bottlenecks at some specific nodes
- These systems may have performance problems
  - High or erratic latency
  - Caused by complex system interactions

- Isolating performance bottlenecks is hard
  - We cannot always examine or modify system components
- We need tools to infer where bottlenecks are
  - Choose which black boxes to open

Example multi-tier system



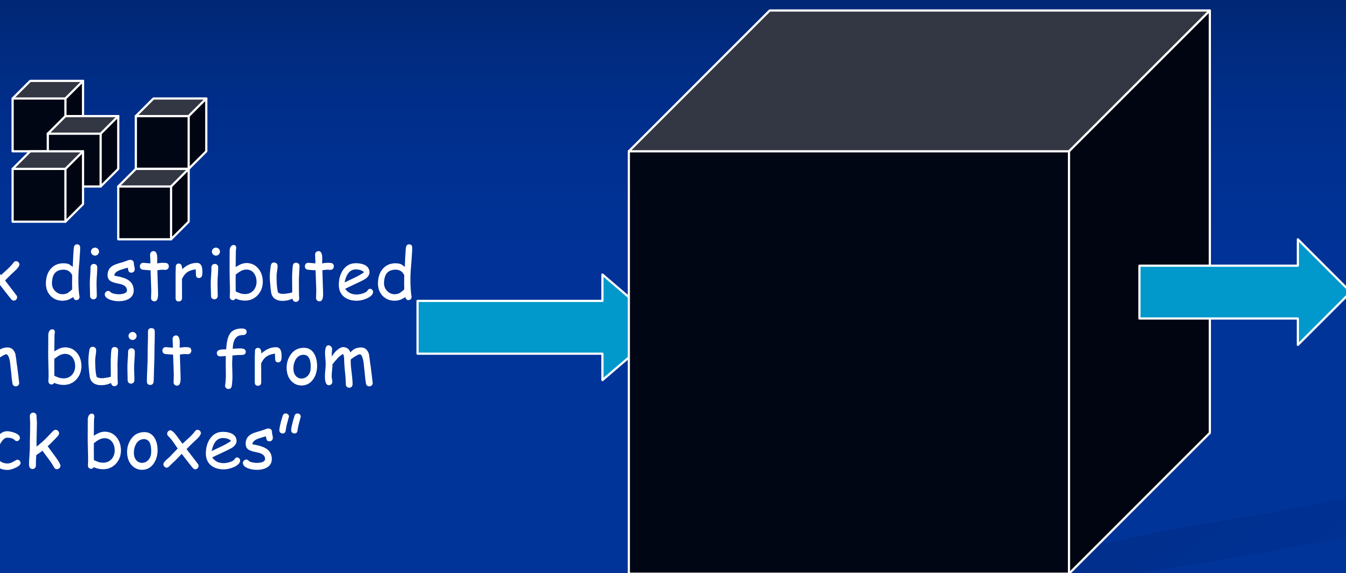
Goals

- Isolating performance bottlenecks
  - Find high-impact causal path patterns
    - Causal path: series of nodes that sent/received messages. Each message is caused by receipt of previous message, and Some causal paths occur many times
    - High-impact: occurs frequently, and contributes significantly to overall latency
  - Identify high-latency nodes on high-impact patterns
    - Add significant latency to these patterns

Then What should We do?

------- Messages Trace is enough

The Black Box



- Desired properties
  - Zero-knowledge, zero-instrumentation, zero-perturbation
  - Scalability
  - Accuracy
  - Efficiency (time and space)

Outline

- Problem statement & goals
- Overview of our approach
- Algorithms
  - The nesting algorithm (RPC)
  - The convolution algorithm (RPC or free-form)
- Experimental results
- Visualization GUI
- Related work
- Conclusions

Overview of Approach

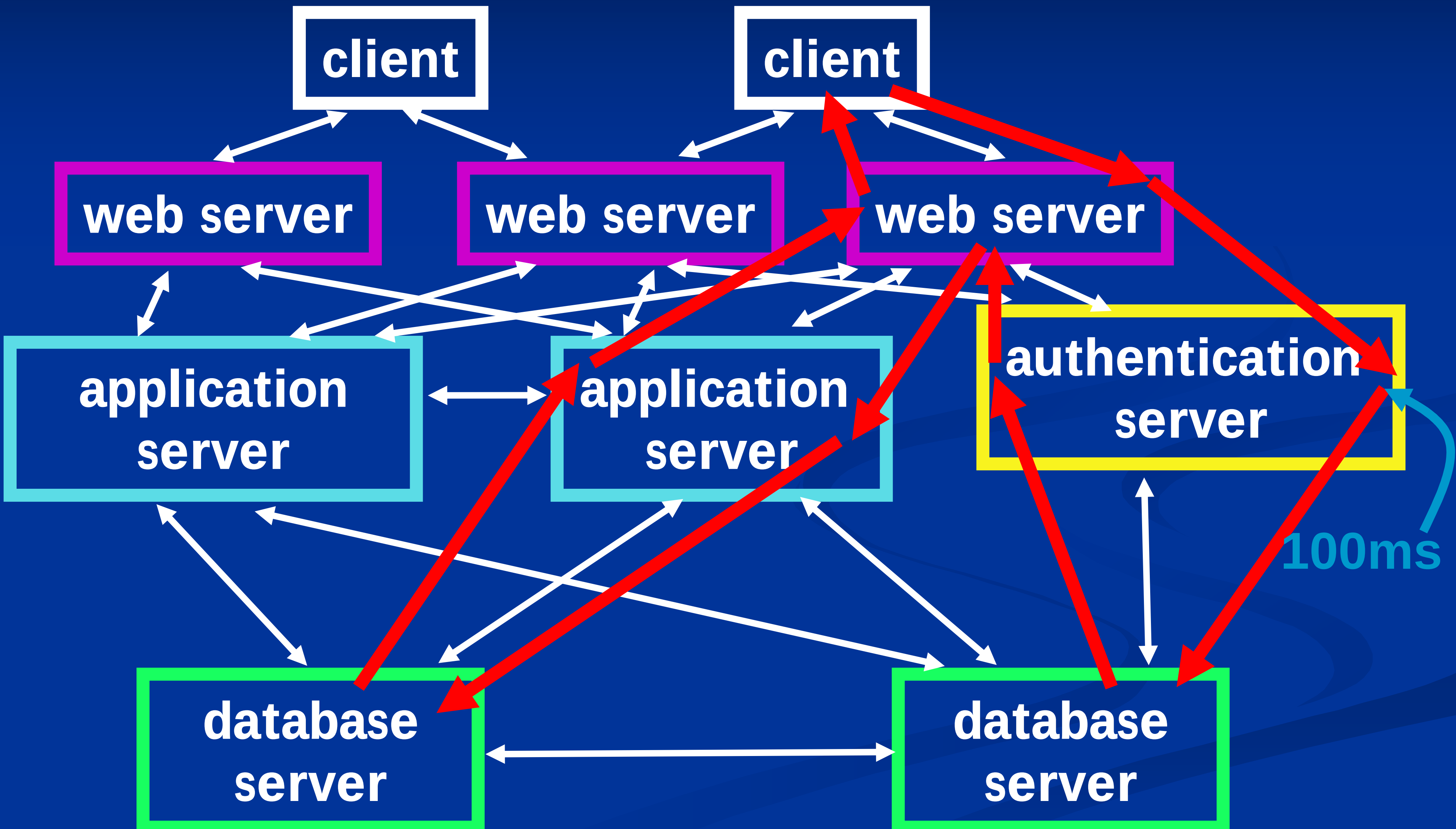
- Obtain traces of messages between components
  - Ethernet packets, middleware messages, etc.
  - Collect traces as non-invasively as possible
  - Require very little information:

    [timestamp, source, destination, *call/return*, *call-id*]

- 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

Recap. causal path



Challenges

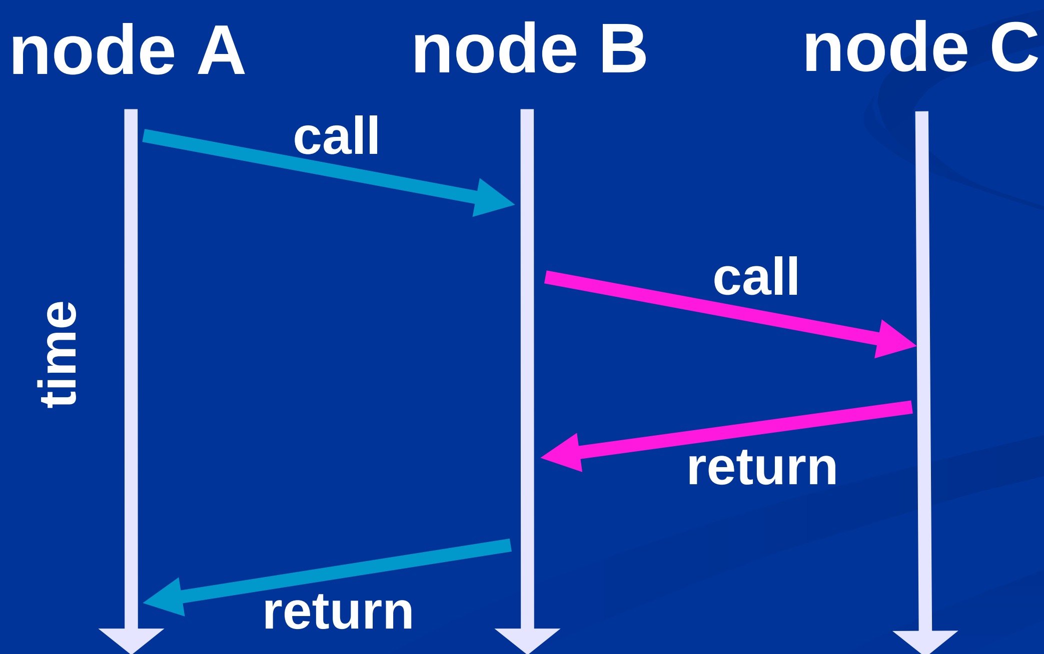
- Trace contain interleaved messages from many causal paths
  - How to identify causal paths?
    - *Causality trace by Timestamp*
- Want only statistically significant causal paths
  - How to differentiate significance?
    - *It is easy! They appear repeatedly*

Outline

- Problem statement & goals
- Overview of our approach
- Algorithms
  - The nesting algorithm (RPC)
  - The convolution algorithm (RPC or free-form)
- Experimental results
- Visualization GUI
- Related work
- Conclusions

The nesting algorithm

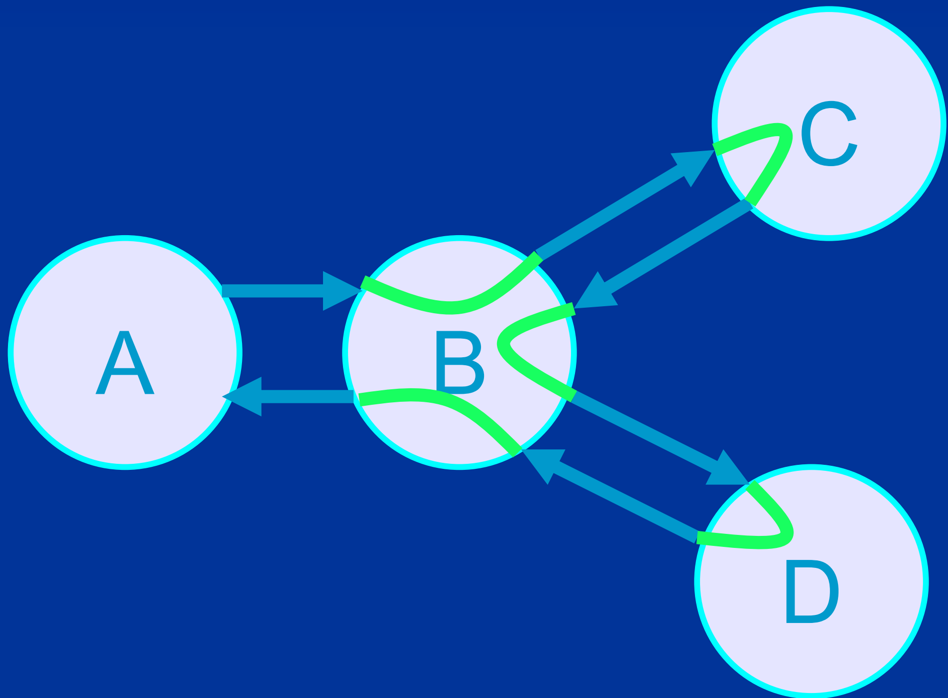
- Depends on RPC-style communication
- Infers causality from "nesting" relationships by message timestamps
  - Suppose A calls B and B calls C before returning to A
  - Then the B↔C call is "nested" in the A↔B call
- Uses statistical correlation

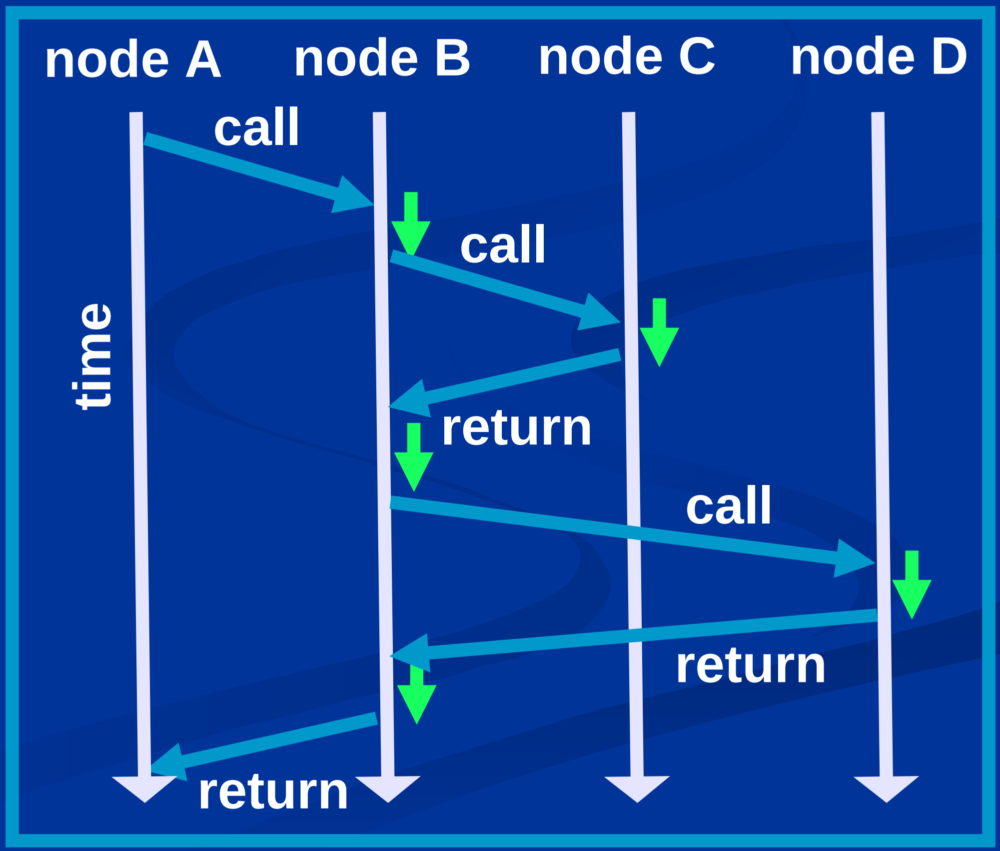


Nesting: an example causal path

Consider this system of 4 nodes

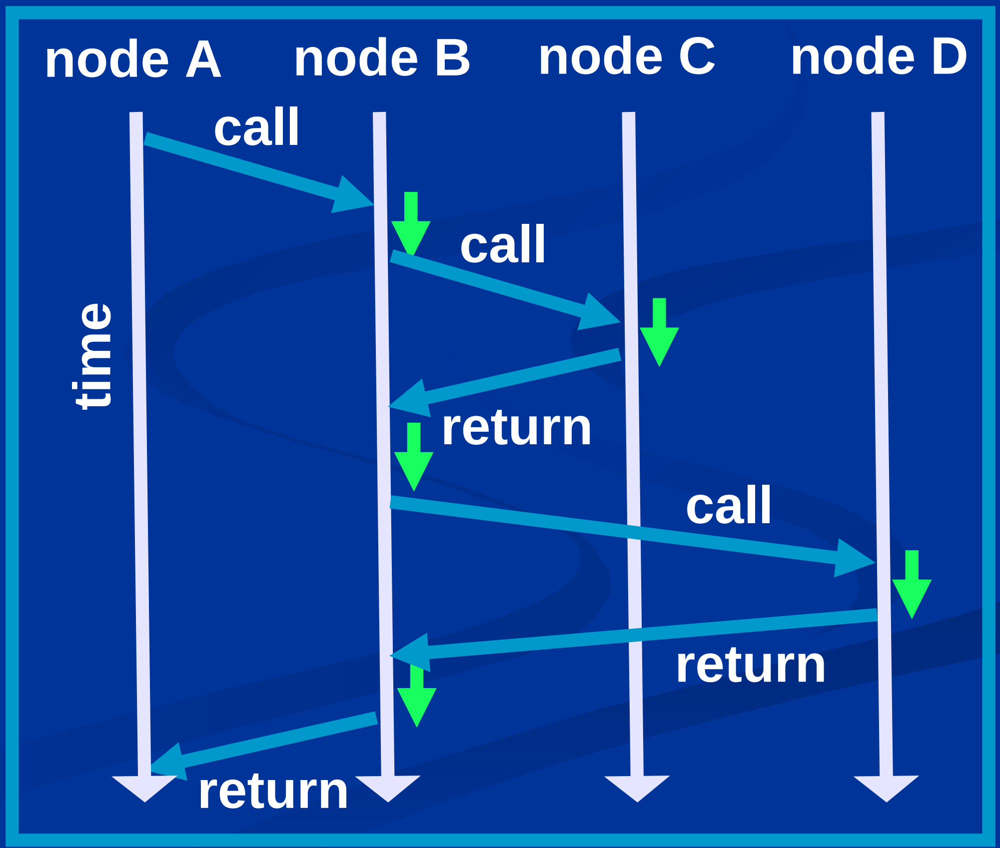
Looking for internal delays at each node





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

Pseudo-code for the nesting algorithm

- Detects calls pairs and find all possible nestings of one call pair in another
- P… the most likely candidate for the causing call for eac…

```
procedure FindCallPairs
for each trace e…
   case CA…
      store…
   case RE…
      find r…
      if ma…
         ren…
         upd…
      add…
      entr…
```

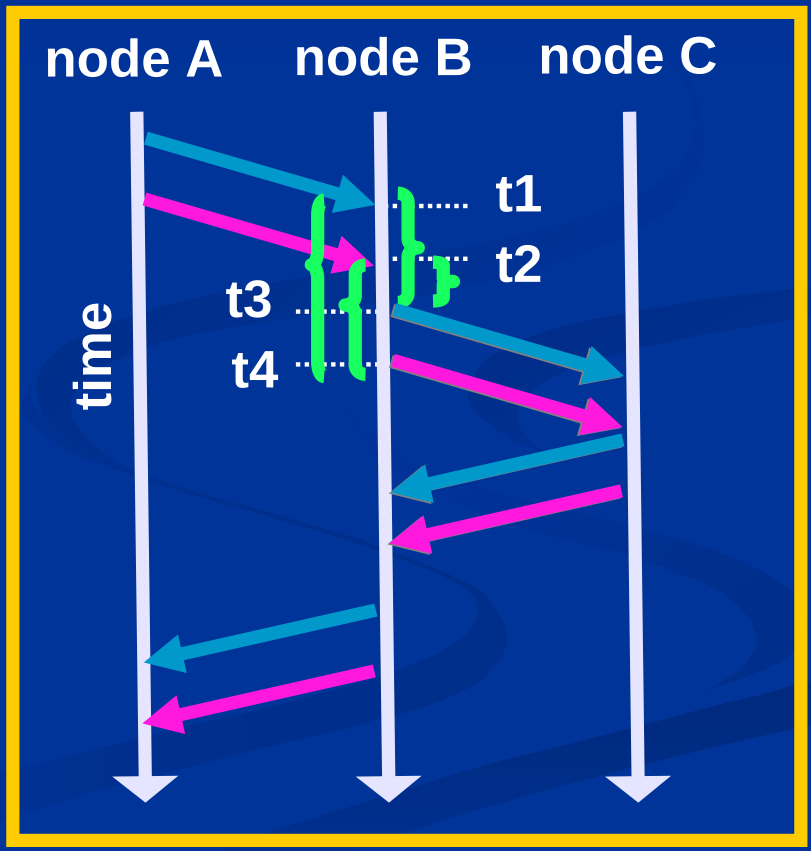
```
procedure ScoreNestings…
for each child (B, C, t…
  for each parent (A, …
    scoreboard[A, B, …

procedure FindNeste…
for each child (B; C; …
   maxscore := 0
   for each p (A, B, t…
      score[p] := scor…
      if (score[p] > m…
         maxscore := s…
         parent := p
   parent.children := p…
```

```
procedure FindCallPaths
initialize hash table Tpaths
for each callpair (A, B, t1, t2)
  if callpair.parents = null then
     root := { CreatePathNode(callpair, t1) }
     if root is in Tpaths then update its latencies
     else add root to Tpaths
function CreatePathNode(callpair (A, B, t1, t4), tp)
   node := new node with name B
   node.latency := t4 - t1
   node.call_delay := t1 - tp
   for each child in callpair.children
     node.edges := node.edges U { CreatePathNode(child, t1)}
   return node
```

Inferring nesting

- An example of Parallel calls
  - Local info not enough
  - Use aggregate info
  - Histograms keep track of possible latencies
    - Medium-length delay will be selected
  - Assign nesting
    - Heuristic methods

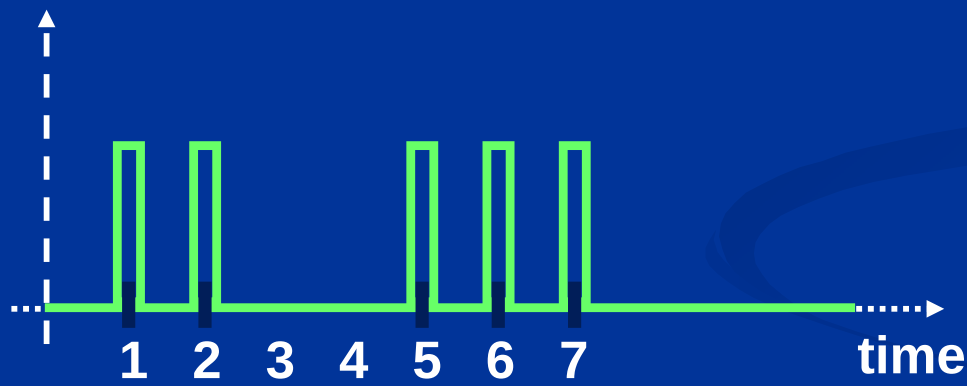


Outline

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

The convolution algorithm

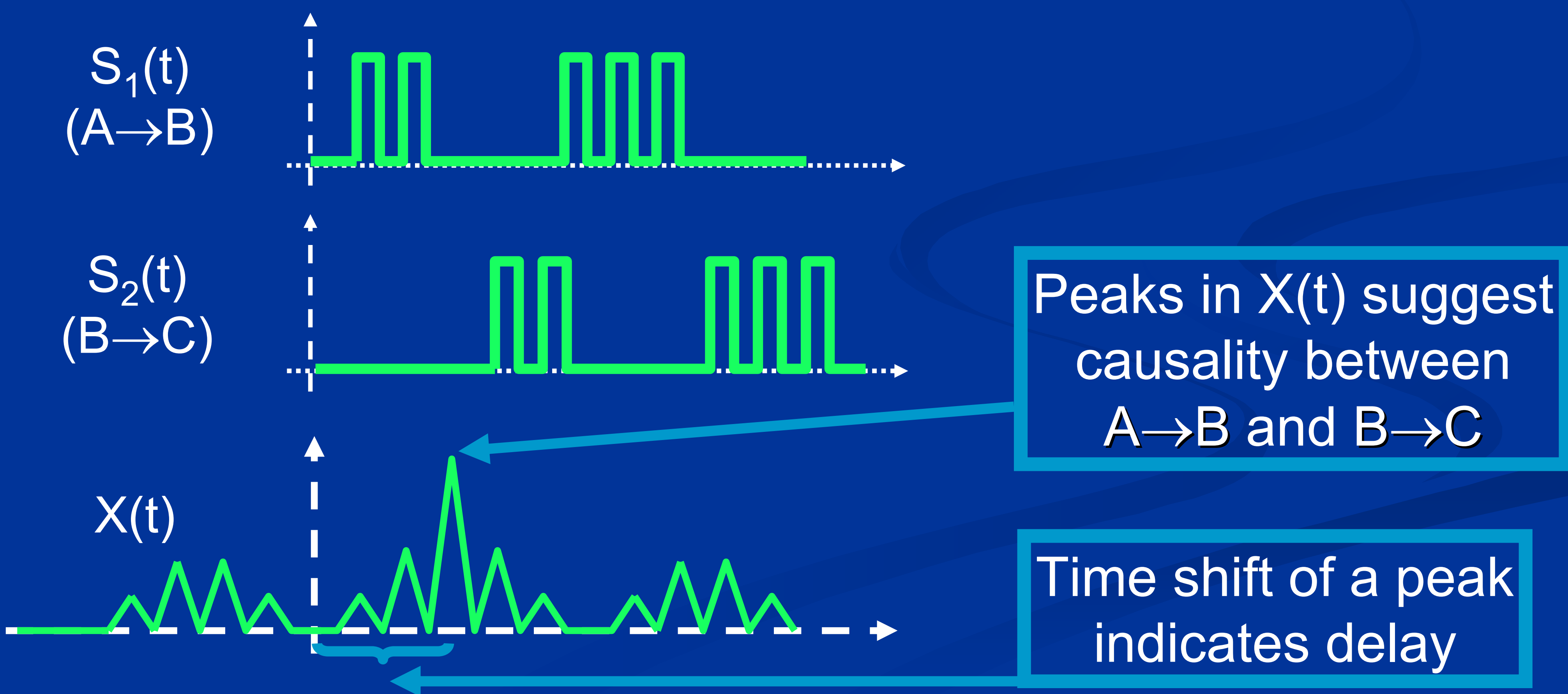
- "Time signal" of messages for each
  <source node, destination node>
  - A sent message to B at times 1,2,5,6,7



S1(t)= A→B messages

The convolution algorithm

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



Convolution details

- Time complexity: O(em+eVlogV)
  - m = messages
  - e = output edges
  - V = 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

Summarize

	Nesting Algorithm	Convolution Algorithm
Communication style	RPC only	RPC or free-form messages
Rare events	Yes, but hard	No
Level of Trace detail	<timestamp, sender, receiver> + call/return tag	<timestamp, sender, receiver>
Time and space complexity	Linear space Linear time	Linear space Polynomial time
Visualization	RPC call and return combined ✹ More compact	Less compact

Outline

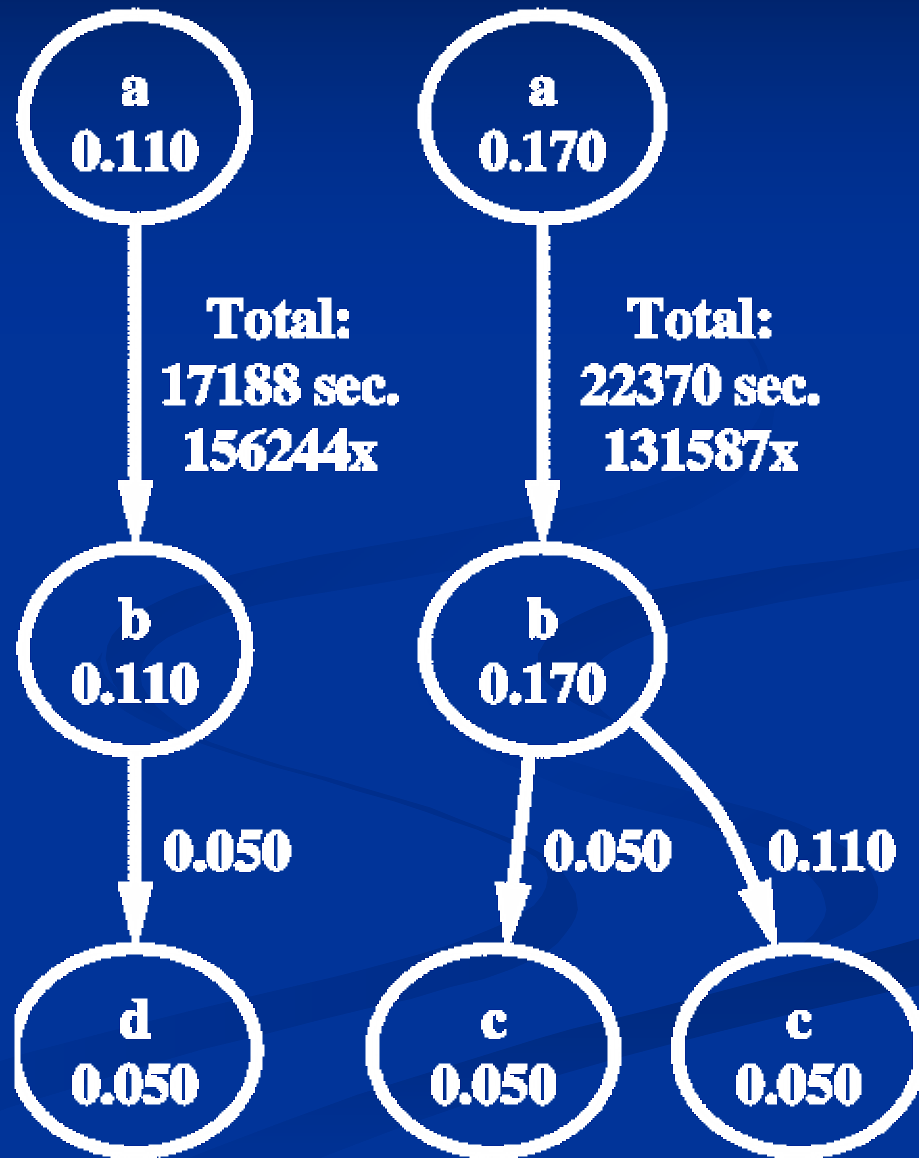
- Problem statement & goals
- Overview of our approach
- Algorithms
- Experimental results
  - Maketrace: a trace generator
  - Maketrace web server simulation
  - Pet Store EJB traces
  - Execution costs
- Visualization GUI
- Related work
- Conclusions

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

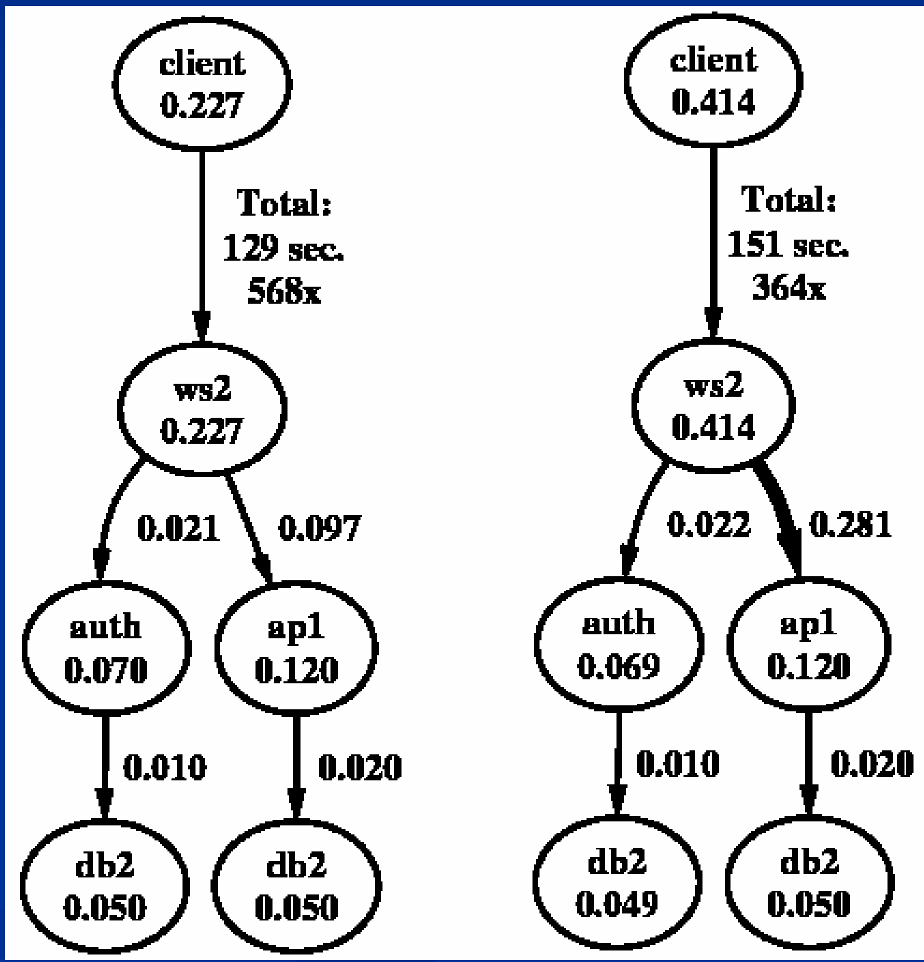
Desired results for one trace

- 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



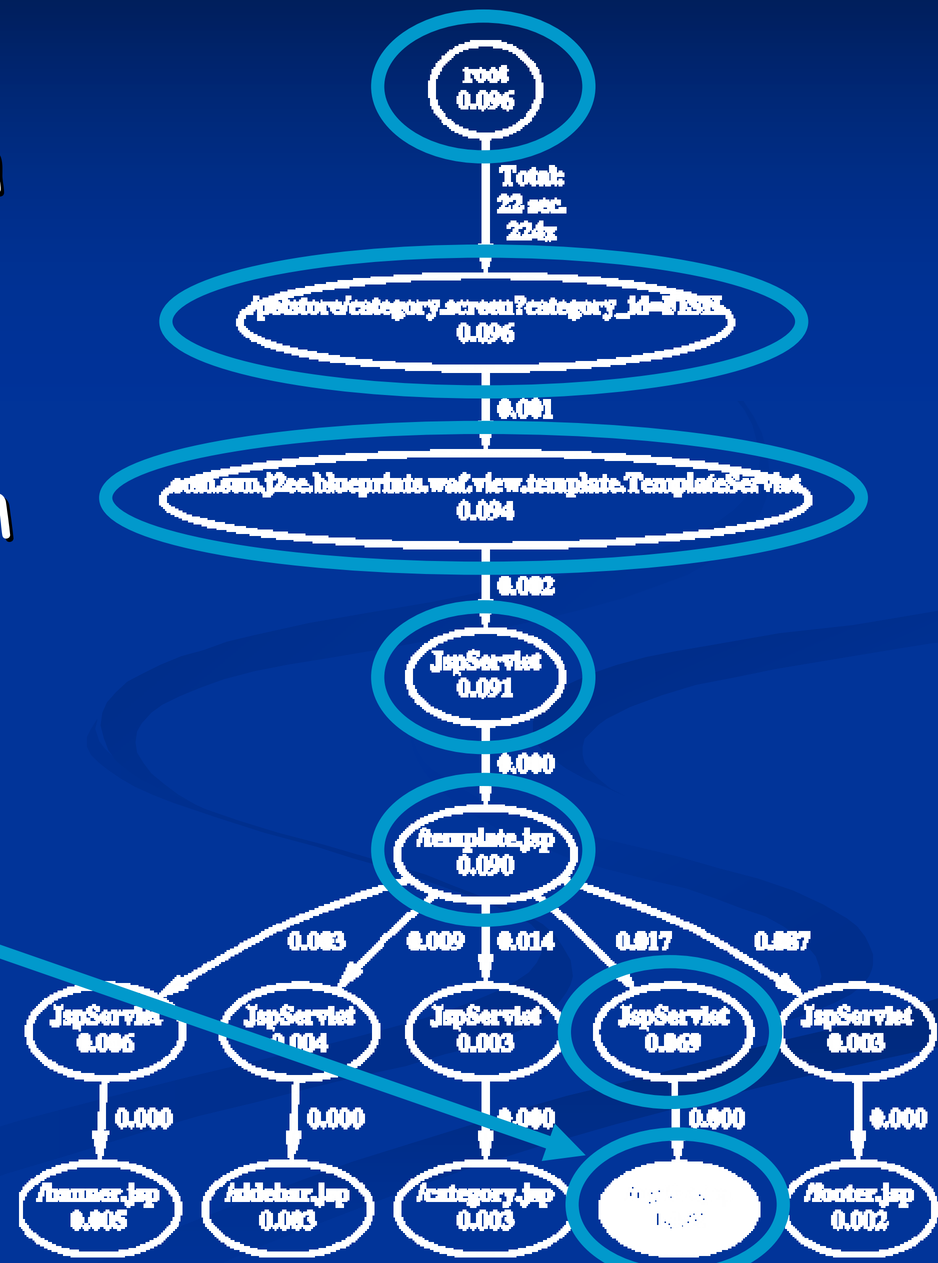
Measuring Added Delay

- Added 200msec delay in WS2
- The nesting algorithm detects the added delay, and so does the convolution algorithm



Results: Petstore

- Sample EJB application
- J2EE middleware for Java
  - Instrumentation from Stanford's PinPoint project
- 50msec delay added in mylist.jsp



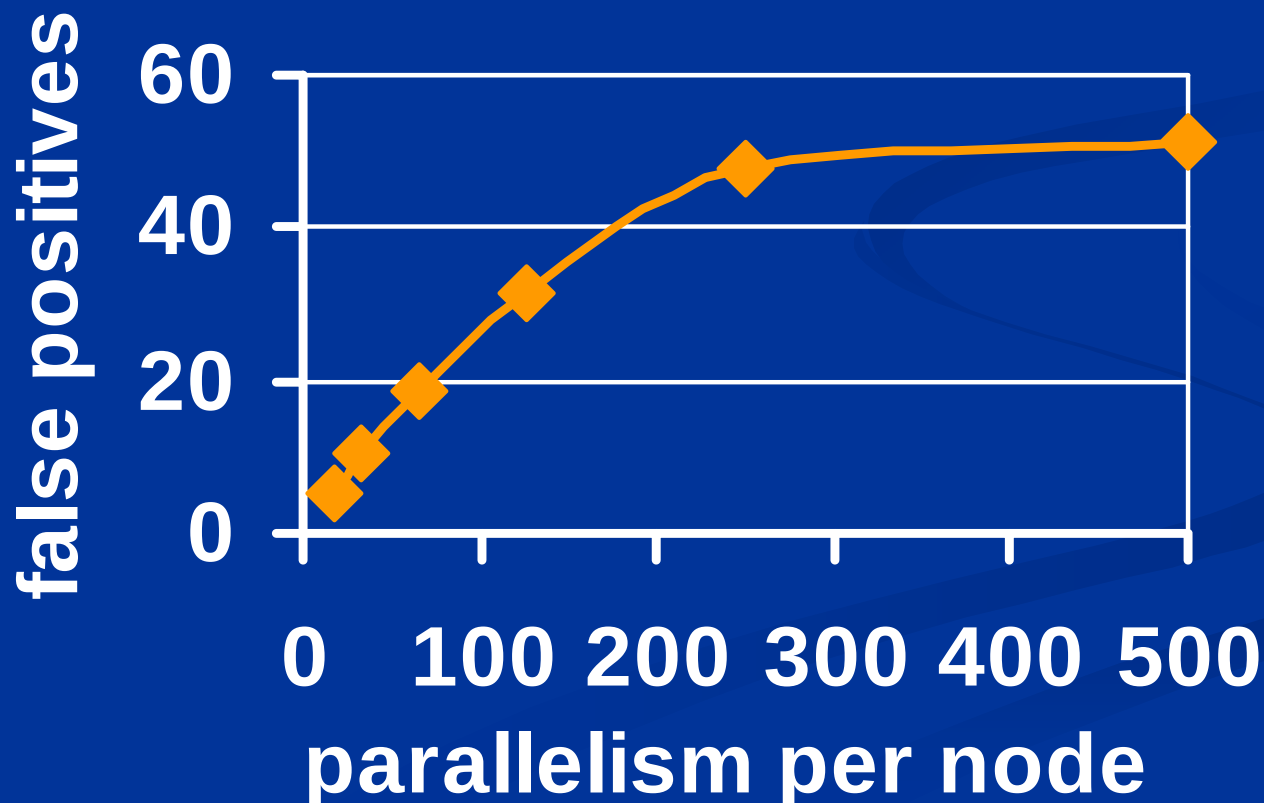
Results: running time

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

More details and results in paper

Accuracy vs. parallelism

- Increased parallelism degrades accuracy slightly
- Parallelism is number of paths active at same time

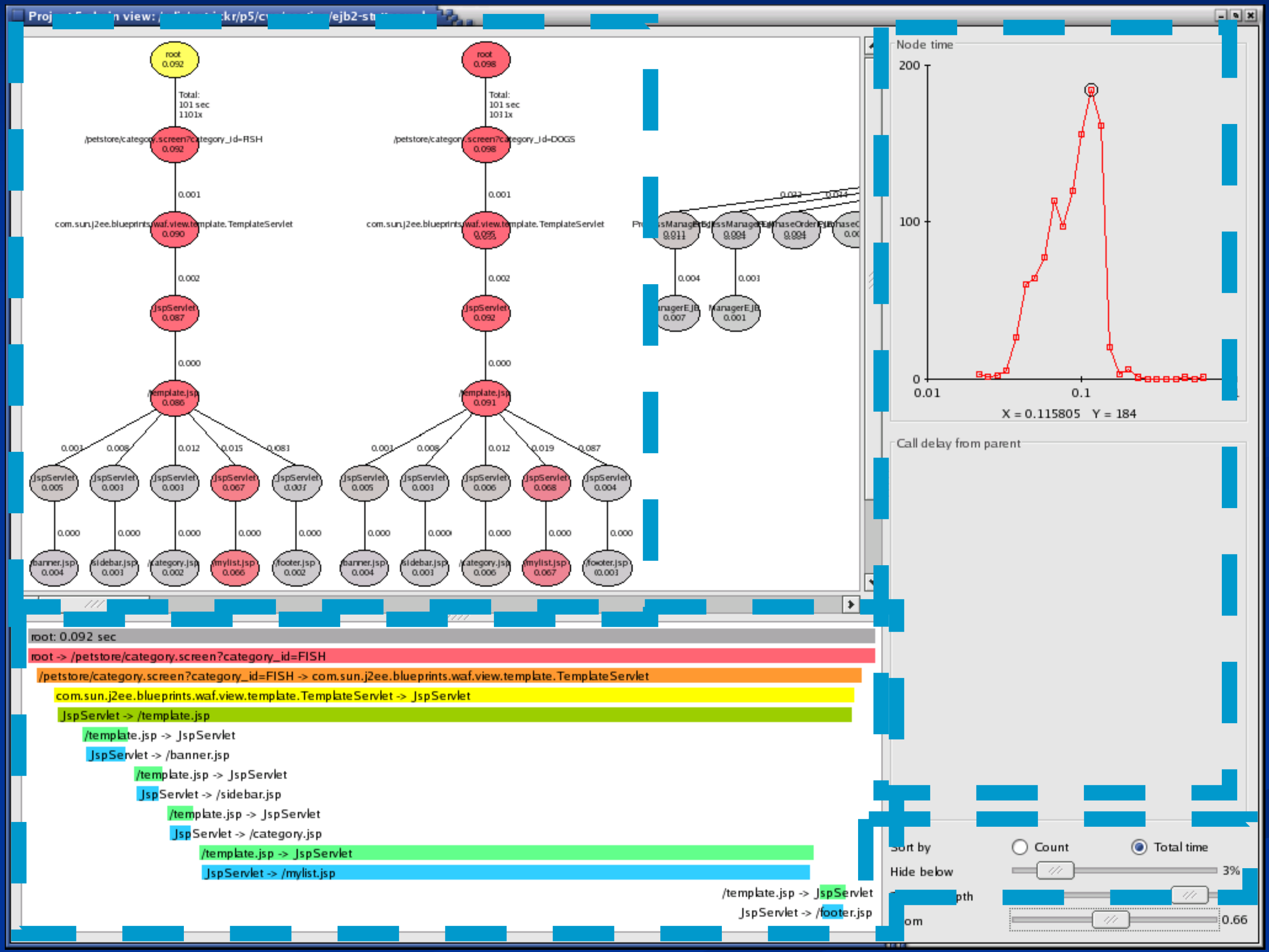


Other results for nesting algorithm

- Clock skew
  - Little effect on accuracy with skew $\leqslant$ delays of interest
- Drop rate
  - Little effect on accuracy with drop rates $\leqslant$ 5%
- Delay variance
  - Robust to $\leqslant$ 30% variance
- Noise in the trace
  - Only matters if same nodes send noise
  - Little effect on accuracy with $\leqslant$ 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

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