Towards Automatic Inference of Task Hierarchies in Complex Systems

Haohui Mai° Chongnan Gao† Xuezheng Liu‡ Xi Wang§ Geoffrey M. Voelker*

University of Illinois at Urbana-Champaign° MIT CSAIL§
Microsoft Research Asia‡ Tsinghua University†
University of California, San Diego*

December 7, 2008
Motivation

- System models are valuable
  - Visualize the design and the implementation
  - Understand the structures of components and their dependency
  - Present dependability measures in an intuitive way
  - Reason and verify the system
- Developers can represent the system as a hierarchical task model
  - Encapsulate implementation details with high-level tasks
  - Allow developers to address dependability problems at various task granularities
Our work

- Explored how well the hierarchical task models can be automatically inferred
  - With minimal or no manual assistance
- Designed and Implemented Scalpel to automatically infer hierarchical task models in complex systems
- Applied Scalpel to two systems
  - Apache HTTP Server
  - PacificA distributed storage system [Lin et al., 2008]
- Encouraging results
Challenges

- All should be done *automatically* in complex systems
- Identify appropriate task boundaries
- Associate dependencies among tasks correctly
- Recover the hierarchical structure among tasks
How it works

Collecting Execution Traces \rightarrow Identifying Leaf Tasks

Inferring Hierarchical Structure \leftarrow Constructing Causal Graph
Collecting Execution Traces

- Trace down calls and their parameters of
  - Synchronization primitives (signal and wait)
  - Socket communication (send and recv)
- Leverage library-based record & replay tool named R2 [Guo et al., 2008] in our implementation
Identifying Leaf Tasks

- Leaf task: smallest unit of work in a task model
- Paritition the execution traces with synchronization points
- Synchronization point: where two threads synchronize their execution and establish a happens-before relation

Rationale
- Dependency only occurs at boundaries
- Relatively independent and self-contained

```
Recv(commit_ack);  
...  
++seq_number;  
...  
```

```
Wait(queue_lock);  
enqueue(element);  
...  
SetEvent(qevent);  
...  
```
Constructing Causal Graph

- Use happens-before relation to infer causal dependency
- Distinguish causal dependency and occasional “run-after” relation
  - Producer - Consumer
  - Mutual exclusion
- Heuristics
  - OS-provided queues (I/O completion ports)
  - Notification mechanisms (events)
  - Efficient to catch shared queues

```c
Wait(write_lock);
...
saveToDisk(data);
...
Send(commit_ack);
...
Recv(commit_ack);
...
++seq_number;
...```
Inferring Hierarchical Structure

- Idea: Identifying frequent patterns in causal graph
- Replace frequent patterns with “super nodes” recursively
- Identifying frequent patterns
  - Canonize sub graph and serialize it deterministically
  - Use hash functions for exact matching
Case Study

- Effectiveness of the task models for debugging
- Effectiveness of capturing developers' intuition
- All experiments on machines with 2.0 GHz Xeon dual-core CPUs, 4 GB memory, running Windows Server 2003 Service Pack 2, and interconnected via a 1 Gb switch
Case Study: Performance Bug in PacificA (I)

- Performance is not satisfactory under stress tests
- Task level profiling based on inferred task models
- Use a top-down approach to identify the problem
  - Use a profiler to collect performance numbers
    - Latency
    - Network bandwidth
    - CPU cycles
- Aggregate profiling data in a per-task manner for each layer
Case Study: Performance Bug in PacificA (II)

- The committing task could not saturate network bandwidth, while at the same time the CPU usage remained low.
Sender threads will block at a call to `sleep()` for 1 second

```c
int Session::WSASendPacket(NetworkStream * pkt) {
    CAutoLock guard(_send_lock);
    while (_send_size > (64 << 20)) // 64 MB
        Sleep(1000);
    ...
    int rt = WSASend(_socket, buf, buf_num, &bytes, 0, (OVERLAPPED*)ce, 0);
    ...
}
```
Case Study: Performance Bug in PacificA (IV)

- Root cause: there is no flow control mechanism at RPC layer when it uses asynchronous communication
  - Thread sends messages in a non-blocking fashion
  - Network layer blocks the thread when the internal buffer is full
  - Threads will all be blocked by the network layer synchronously at high workload

- Caused by poor interactions across software layers

- A clear hierarchical model helped developers to identify the location of the bug and also understand its root cause
Case Study: Task Model of Apache HTTP Server

- Successfully capture the Apache service cycle for SVN checkout operations
### Case Study: Statistics

<table>
<thead>
<tr>
<th></th>
<th>Apache</th>
<th>PacificA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SLOC</strong></td>
<td>819676</td>
<td>54458</td>
</tr>
<tr>
<td><strong>Leaf Tasks</strong></td>
<td>423952</td>
<td>10636</td>
</tr>
<tr>
<td><strong>Events</strong></td>
<td>0</td>
<td>47</td>
</tr>
<tr>
<td><strong>IOCP</strong></td>
<td>23</td>
<td>16</td>
</tr>
<tr>
<td><strong>Socket</strong></td>
<td>527</td>
<td>77</td>
</tr>
<tr>
<td><strong>Mutex</strong></td>
<td>210472</td>
<td>4950</td>
</tr>
<tr>
<td><strong>Same thread</strong></td>
<td>193972</td>
<td>11304</td>
</tr>
<tr>
<td><strong>Running Time: Extracting Task Models</strong></td>
<td>5.95s</td>
<td>32.02s</td>
</tr>
<tr>
<td><strong>Running Time: Native run</strong></td>
<td>9.66s</td>
<td>20.79s</td>
</tr>
<tr>
<td><strong>Running Time: Execution Time</strong></td>
<td>10.00s</td>
<td>28.36s</td>
</tr>
<tr>
<td><strong>Overhead</strong></td>
<td>3.52%</td>
<td>36.41%</td>
</tr>
</tbody>
</table>
Conclusion & Future Work

- Hierarchical task models of complex systems can be inferred with few or no annotations

- Future work
  - Extend the trace collecting method to collect memory operations
  - More effective heuristics to prune “run-after” cases
  - Experiment more graph mining algorithm for recovering task hierarchies
  - Evaluate more systems
Acknowledgment

- Our colleague Wenguang Chen for valuable feedback and discussions
- Zhenyu Guo for the R2 work
- Wei Lin and other PacificA developers
- Support from System Research Group in Microsoft Asia and High Performance Computing Group in Tsinghua University

Collect Execution Traces and Identify Leaf Tasks

```
winnt_accept() {
    while (...) {
        ...mpm_get_completion_context();
        Wait(qlock);
        ...Wait(qwait_event);
        ...AcceptEx(conn);
        Wait(conn);
        ...PostQueue(ThrDispatch);
    }
}

worker_main() {
    while (...) {
        ...winnt_get_connection();
        mpm_recycle_completion_context();
        Wait(qlock);
        SetEvent(qwait_event);
        ...GetQueue(ThrDispatch);
        ...ap_process_connection();
        ap_core_input_filter();
        svn_io_read();
        svn_io_getc();
        ap_lingering_close();
    }
}
```
Constructing Causal Graph

```c
winnt_accept() {
    while (...) {
        mpm_get_completion_context();
        Wait(qlock);
        Wait(qwait_event);  
        AcceptEx(conn);
        Wait(conn);
        PostQueue(ThrDispatch);
    }
}

worker_main() {
    while (...) {
        ... 
        winnt_get_connection();
        mpm_recycle_completion_context();
        winnt_get_connection();
        SetEvent(qwait_event);
        GetQueue(ThrDispatch);
        ap_process_connection();
        ap_core_input_filter();
        svn_io_read();
        svn_io_getc();
        ap_lingering_close();
    }
}
```
Inferring Hierarchical Structure

```
winnt_accept() {
    while (...) {
        ...mpm_get_completion_context();
        1
        Wait(qlock);
        2
        Wait(qwait_event);
        3
        AcceptEx(conn);
        4
        Wait(conn);
        5
        PostQueue(ThrDispatch);
    }
}
```

```
worker_main() {
    while (...) {
        5
        winnt_get_connection();
        mpm_recycle_completion_context();
        6
        Wait(qlock);
        SetEvent(qwait_event);
        ...GetQueue(ThrDispatch);
        7
        ap_process_connection();
        8
        ap_core_input_filter();
        svn_io_read();
        9
        svn_io_getc();
        A
        ap_lingering_close();
        B
    }
}
```