Fay: Extensible Distributed Tracing from Kernels to Clusters

Prasanna Gautam

UConn CSE-5095-005 (Sept 23, 2014)

Vision

- Know what clusters are doing at any time
  - Down to machine's system calls
- Ask any question to the cluster
- Collect answers efficiently
  - Be able to run continuously
- Safely run in production without instrumentation
- Apply data mining techniques to clusters

Goals

- Platform should allow arbitrary high-level queries about any aspect of system
- Stateful Probes
- In-machine Data Reduction (filter,aggregate)
- Low overhead tracing

Fay: Features

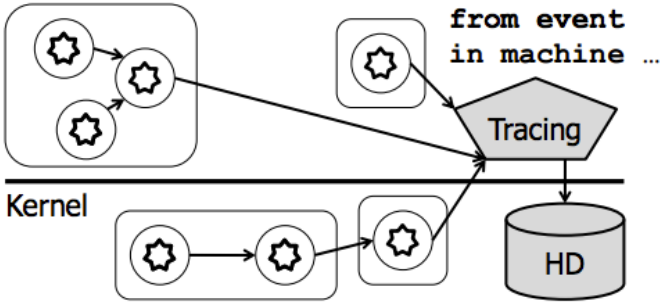
1. Can be applied to existing, live Windows servers
2. Single query
3. Pervasively data-parallel
4. Inline, safe machine-code at tracepoints

Integrating High Level languages

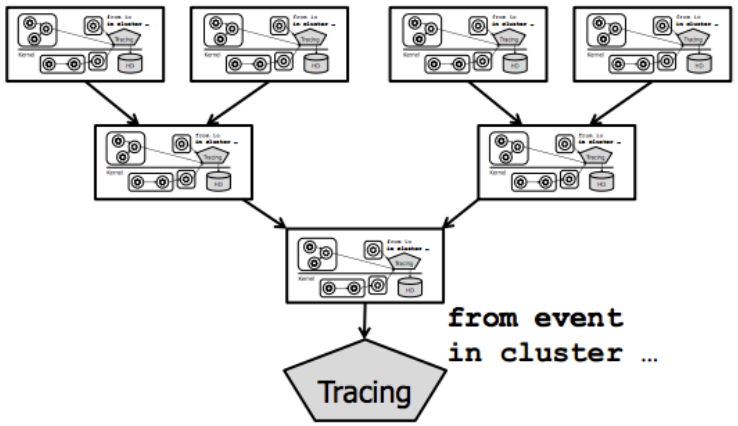
- Extending DryadLINQ to FayLINQ
  - Distributed Querying
  - Hide the distributed nature of queries
  - High level declarative data processing
  - Can simultaneously express
    - Trace Collection
    - Trace Event Analysis
    - Persisting trace event logs
- Powershell interface

Tracing A Cluster

FayLINQ tracing a single machine:



FayLINQ tracing a cluster:



FayLINQ Example:

```
cluster.Function(kernel, "ExAllocate*")
.Count(event => (event.time < Now.AddMinutes(5)));
```

PowerShell Example:

```
# This is a simple comment
function Hello($name) {
  Write-host "Hello $name"
}

function add($left, $right=4) {
    if ($right -ne 4) {
        return $left
    } elseif ($left -eq $null -and $right -eq 2) {
        return 3
    } else {
        return 2
    }
}

$number = 1 + 2;
$number += 3

Write-Host Hello -name "World"

$an_array = @(1, 2, 3)
$a_hash = @{"something" = "something else"}

& notepad .\readme.md
```

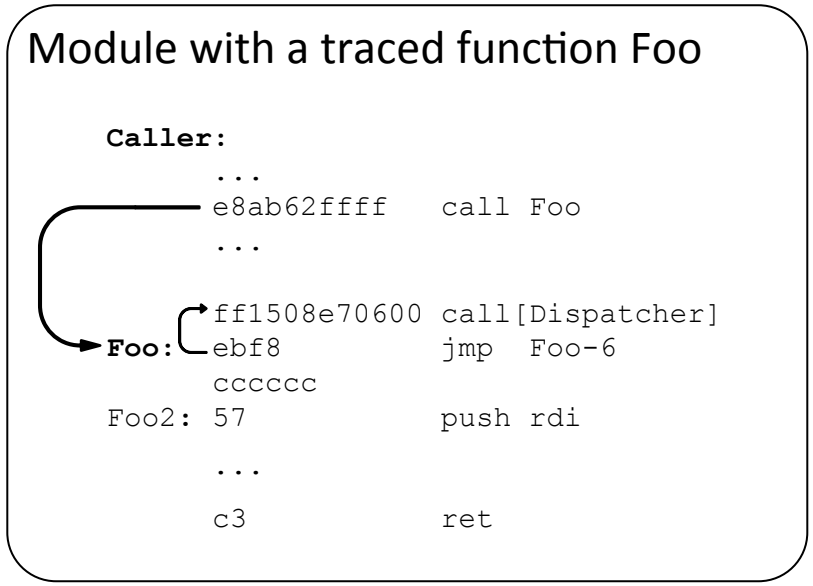
Fay vs Specialized Tracing

- Fay is general, but can efficiently do
  - Tracing across abstractions, systems (Magpie)
  - Predicated and windowed tracing (Streams)
  - Probabilistic tracing (Chopstix)
  - Flight recorders, performance counters, …

Tracing with Fay

- Dynamic Instrumentation
  - Minimally intrusive (only first machine code instruction is changed)
  - Inline invocations
    - Limits to a single process, kernel
    - Each address space is traced separately

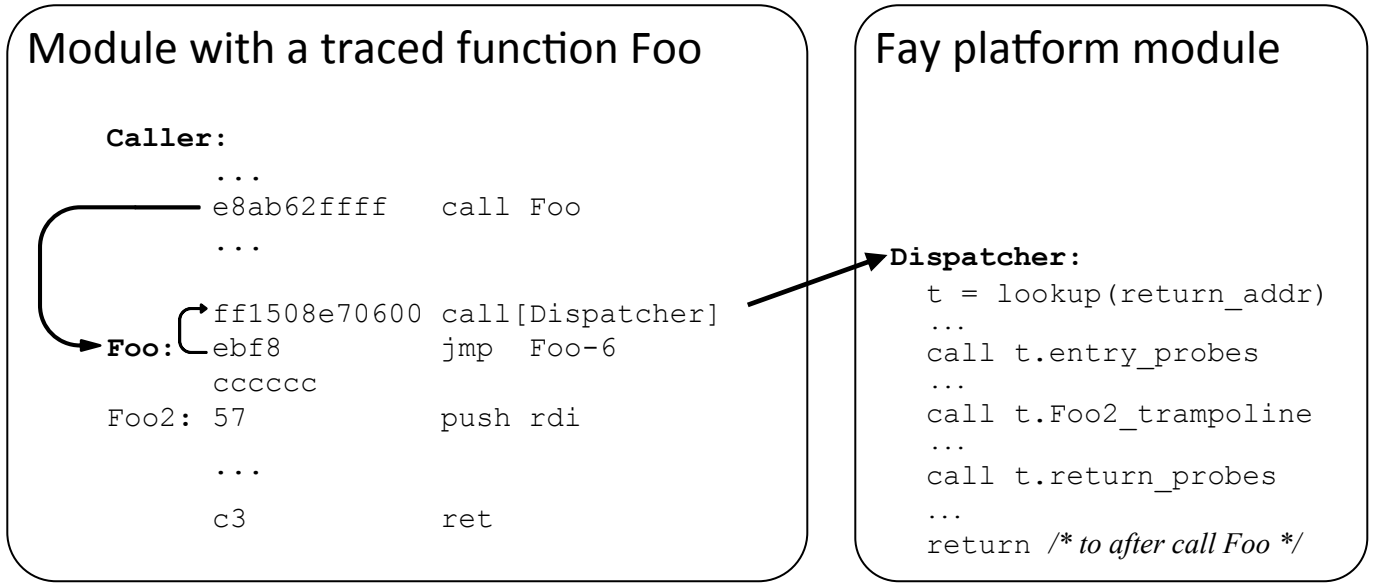
Low-Level Code Instrumentation



- Replace 1st opcode of functions

http://www.sigops.org/sosp/sosp11/current/2011-Cascais/22-erlingsson.pptx

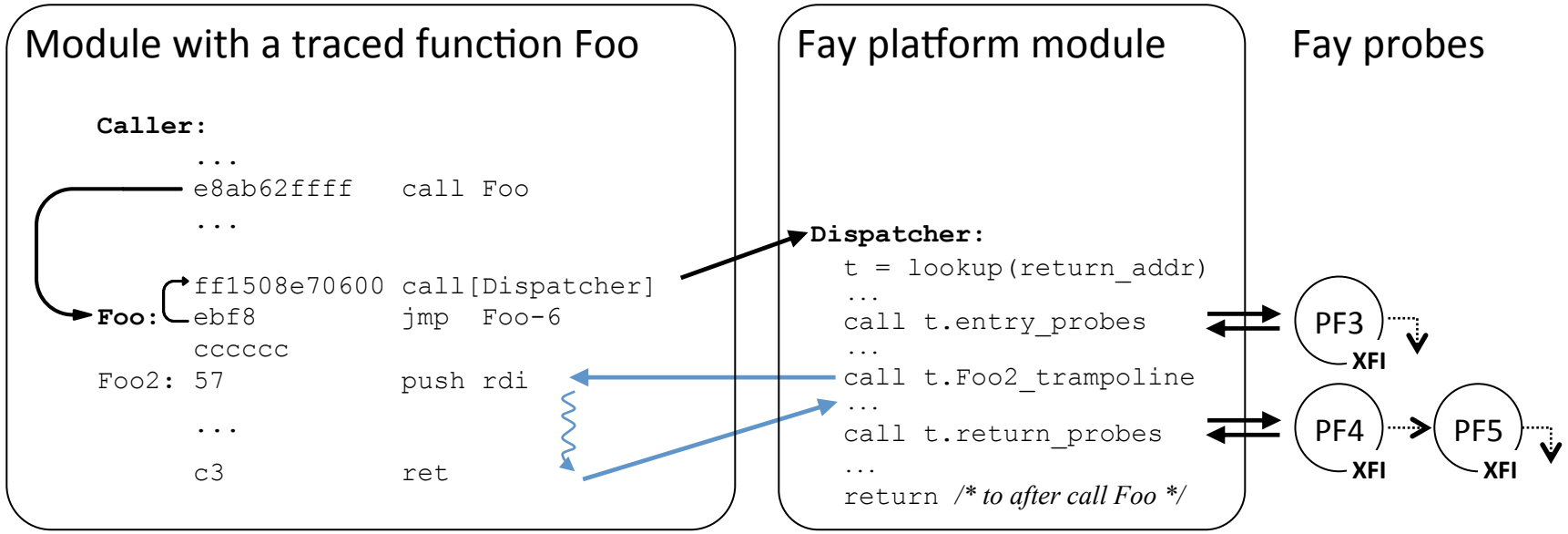
Low-Level Code Instrumentation



- Replace 1st opcode of functions
- Fay dispatcher called via trampoline

http://www.sigops.org/sosp/sosp11/current/2011-Cascais/22-erlingsson.pptx

Low-Level Code Instrumentation

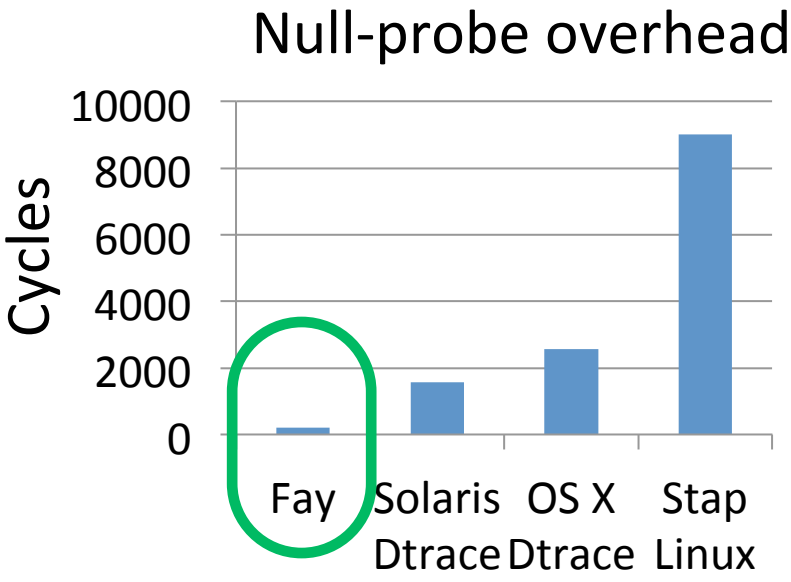


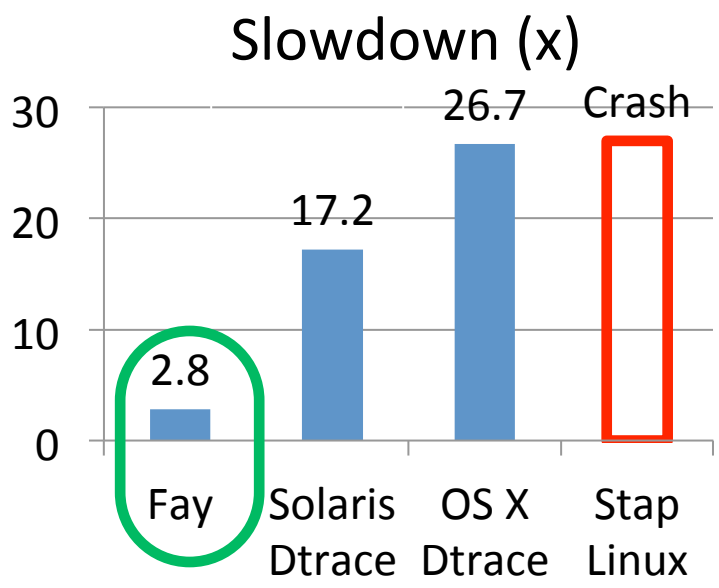
- Replace 1st opcode of functions
- Fay dispatcher called via trampoline
- Fay calls the function, and entry & exit probes

http://www.sigops.org/sosp/sosp11/current/2011-Cascais/22-erlingsson.pptx

Benchmark: Trivial System call in Tight Loop

- Fay adds 220 to 430 cycles per traced function
- Fay adds 180% CPU to trace all kernel functions
- Approx 10x faster than Dtrace, SystemTap





http://www.sigops.org/sosp/sosp11/current/2011-Cascais/22-erlingsson.pptx

Case Study: Performance Diagnosis

- Interactively using Windows Command Shell
  - High frequency of system calls
  - Outputting 16MB ASCII file in minimized console window => 3.75 million system calls

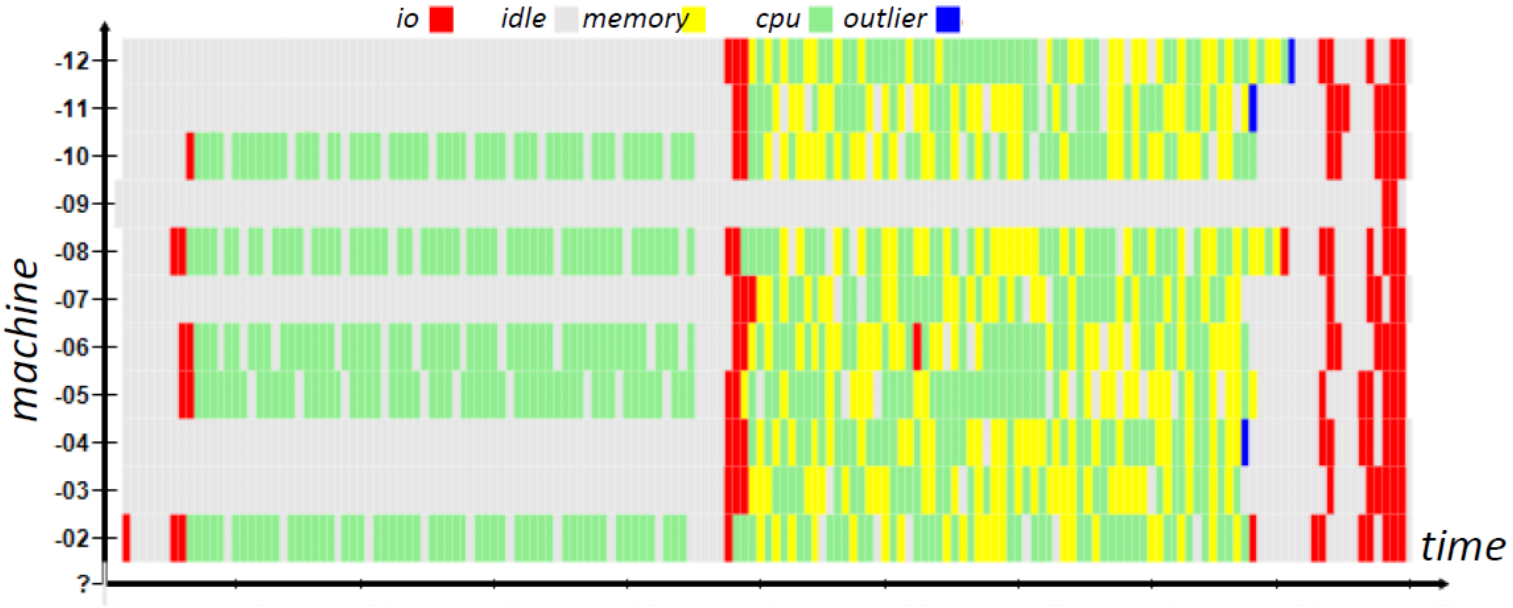
Windows System Call	Count	Callers
`NtRequestWaitReplyPort`	1,515,342	cmd.exe conhost
`NtAlpcSendWaitReceivePort`	764,503	CSRSS
`NtQueryInformationProcess`	758,933	CSRSS
`NtReplyWaitReceivePort`	757,934	conhost

Table 1: The processes in the command shell case study, and a count of how often they made the relevant system calls. The two calling `NtRequestWaitReplyPort` did so about equally often.

Distributed Performance Counters

- Fay tracing can trivially count events across machines in cluster
- User-Controllable and Efficient aggregation
- Per-
  - Process
  - Thread
  - Module
- User-mode and Kernel

Automatic Analysis of Cluster Behavior



- Automatically categorize cluster behavior, based on system call activity
  - Without measurable overhead on the execution
  - Without any special Fay data-mining support

Scalability, Fault Tolerance

- Used Fay to trace all 8,001 hotpatchable functions in Windows kernel and count
  - Worst case scenario
  - Fay call dispatcher creates extra stack
  - Branch-prediction miss on function return
- Dtrace is slower but not really comparable
- Randomly killing processes, threads, machines
  - Thread local state is lost
  - Dryad ensured cluster level fault tolerance

Related Work

- Dynamic Instrumentation
  - New Tracing tools that use dynamic instrumentation like Ftrace
- Safe OS extensions
  - Informer profiler in 1969 did this
  - Google's NaCl runs assembly code in browser
- Declarative Tracing and Debugging
  - Integration with LINQ: PQL, PTQL

Issues

- Only specific to Windows
  - Authors mention easy ports to other Oses
- LINQ is tied to .NET runtime
  - Possibly usable in Mono in other operating systems

Questions