INTROPERF: TRANSPARENT CONTEXT-SENSITIVE MULTI-LAYER PERFORMANCE INFERENCE USING SYSTEM STACK TRACES

Chung Hwan Kim*, Junghwan Rhee, Hui Zhang, Nipun Arora, Guofei Jiang, Xiangyu Zhang*, Dongyan Xu*

NEC Laboratories America
*Purdue University and CERIAS

NEC Laboratories America
Relentless passion for innovation

PURDUE UNIVERSITY

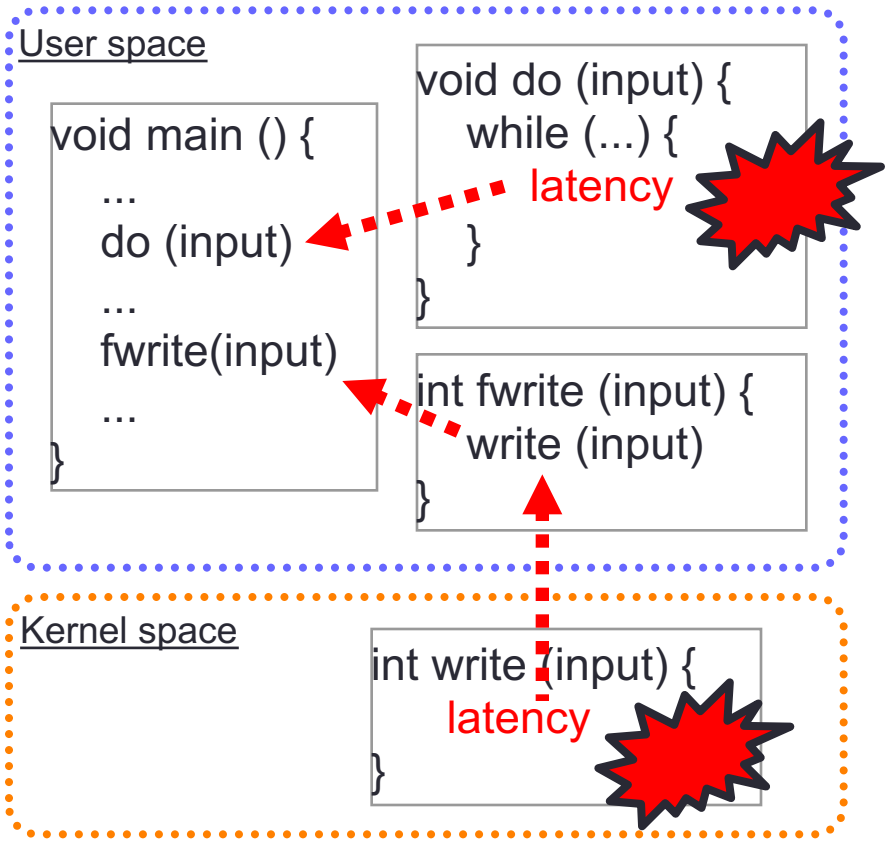
Performance Bugs

- Performance bugs
  - Software defects where relatively simple source-code changes can significantly speed up software, while preserving functionality [Jin et al., PLDI12].
  - Common issues in most software projects and these defects are hard to be optimized by compilers due to software logic.
  - Many performance bugs escape the development stage and cause cost and inconvenience to software users.

Diagnosis of Performance Bugs is Hard

- Diverse root causes
  - Input/workload
  - Configuration
  - Resource
  - Bugs
  - Others
- Performance overhead propagates.

=> Need performance analysis in a global scope!



"Performance problems require understanding all system layers"

-Hauswirth et al., OOPSLA '04

Diagnosis of Performance Bugs

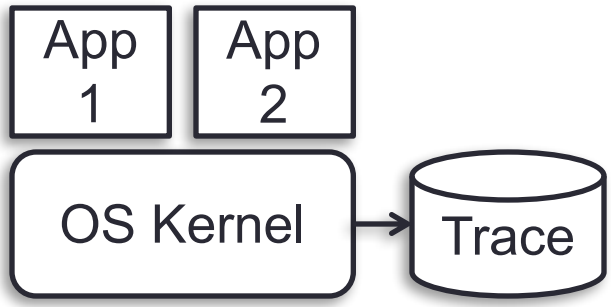
- Development stage
  - Source code is available.
  - Developers have knowledge on programs.
  - Testing workload
  - Heavy-weight tools such as profilers and dynamic binary instrumentation are often tolerable.

- Post-development stage
  - Many users do not have source code.
  - Third-party code and external modules come in binaries.
  - Realistic workload at deployment
  - Low overhead is required for diagnosis tools.

- **Q: How to analyze performance bugs and find their root causes in a post-development stage with low overhead?**

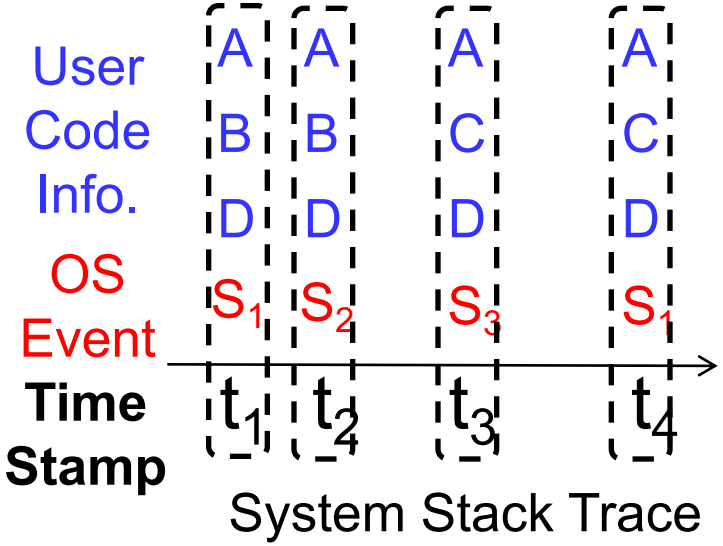
OS Tracers and System Stack Trace

- Many modern OSes provide tracing tools as swiss-army-tools
  - These tools provide tracing of OS events.
  - Examples: SystemTap, Dtrace, Microsoft ETW



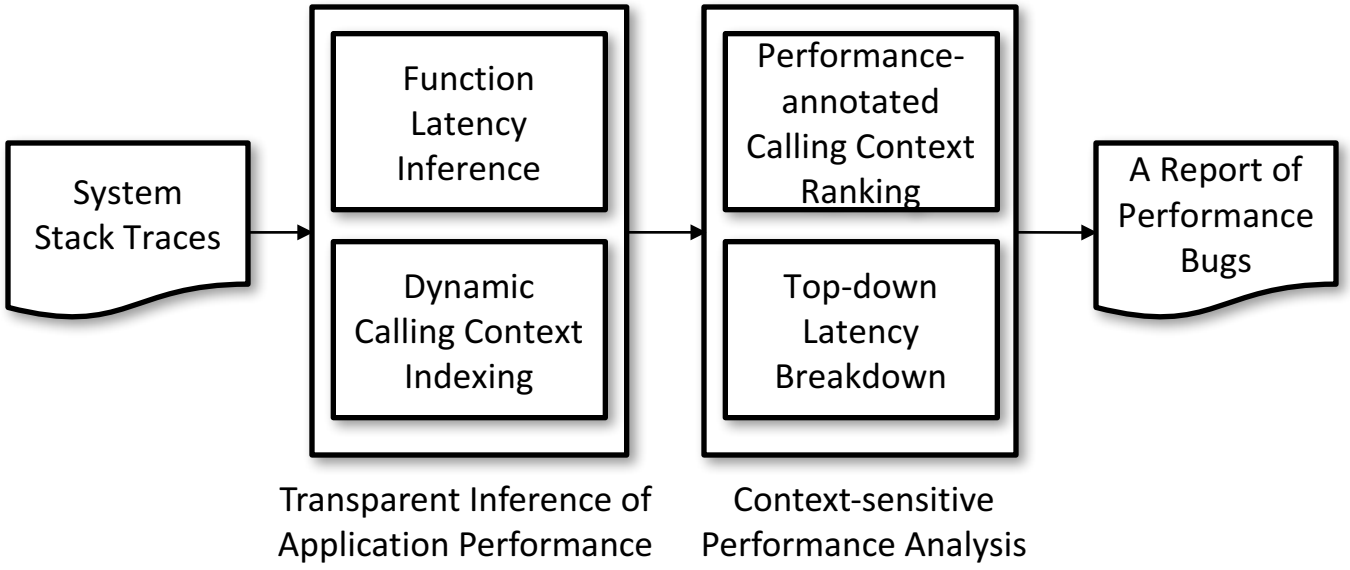
- Advanced OS tracers provide stack traces.
  - We call OS events + stack traces = **system stack traces**.
  - Examples: Microsoft ETW, Dtrace

- Challenges
  - Events occur on OS events.
  - Missing application function latency: **How do we know which program functions are slow?**



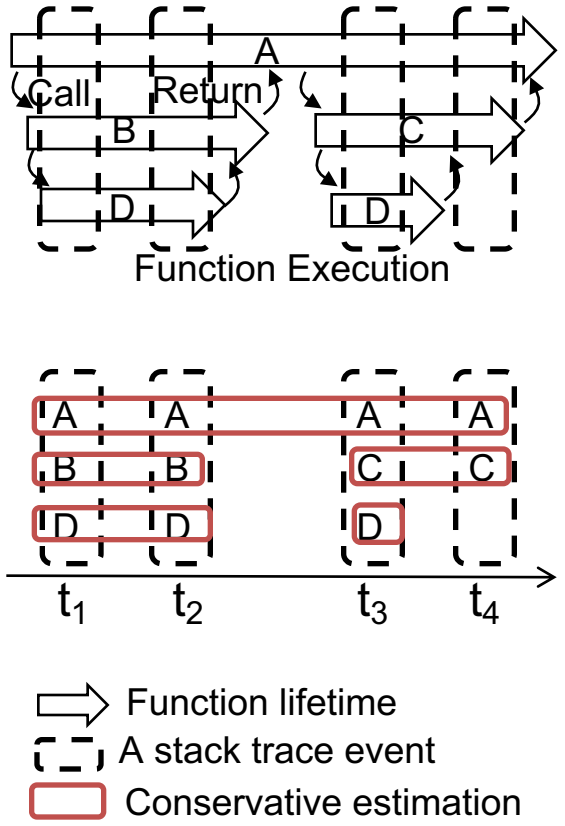
IntroPerf

- **IntroPerf:** A diagnosis tool for **Perf**ormance **Intro**spection based on system stack traces
- **Key Ideas**
  - Function latency inference based on the continuity of a calling context
  - Context sensitive performance analysis



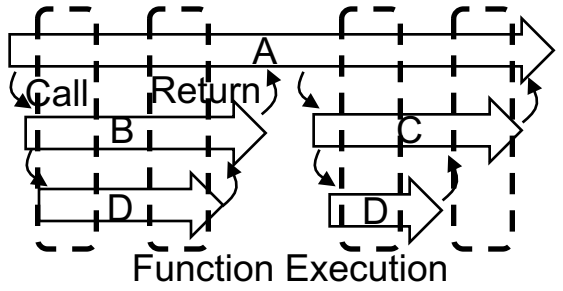
Inference of Function Latencies

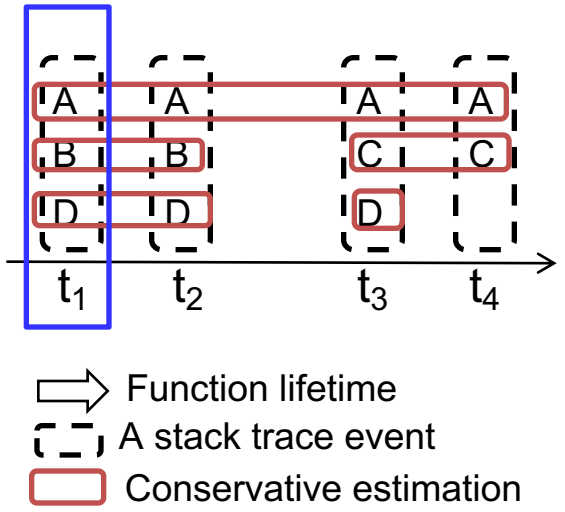
- Inference based on the continuity of a function in the context
  - Algorithm captures a period of a function execution in the call stack without a disruption of its context



Inference of Function Latencies

- Inference based on the continuity of a function in the context
  - Algorithm captures a period of a function execution in the call stack without a disruption of its context



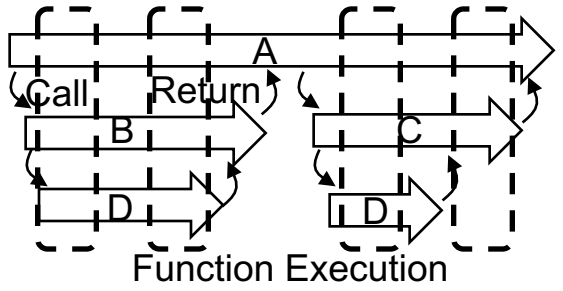


IsNew	ThisStack	Register (Time)
Yes	A	A (T1-T1)
Yes	B	B (T1-T1)
Yes	D	D (T1-T1)

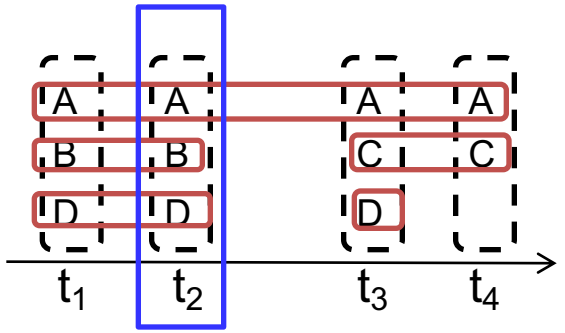
Captured Function Instances

Inference of Function Latencies

- Inference based on the continuity of a function in the context
  - Algorithm captures a period of a function execution in the call stack without a disruption of its context



IsNew	ThisStack	Register (Time)
No	A	A (T1-**T2**)
No	B	B (T1-**T2**)
No	D	D (T1-**T2**)

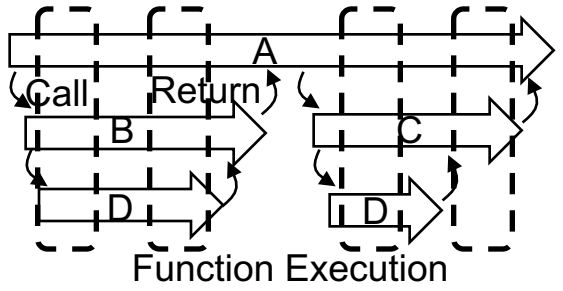


Function lifetime
A stack trace event
Conservative estimation

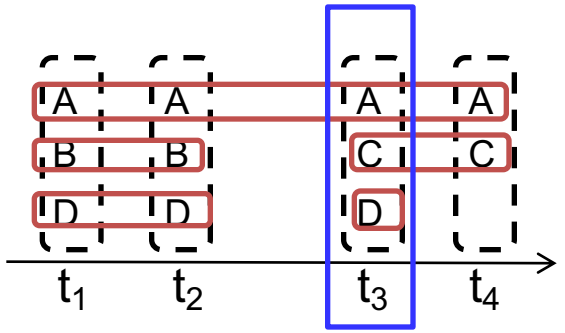
Captured Function Instances

Inference of Function Latencies

- Inference based on the continuity of a function in the context
  - Algorithm captures a period of a function execution in the call stack without a disruption of its context



IsNew	ThisStack	Register (Time)
No	A	A (T1-**T3**)
Yes	**C**	**C (T3-T3)**
Yes	**D**	**D (T3-T3)**



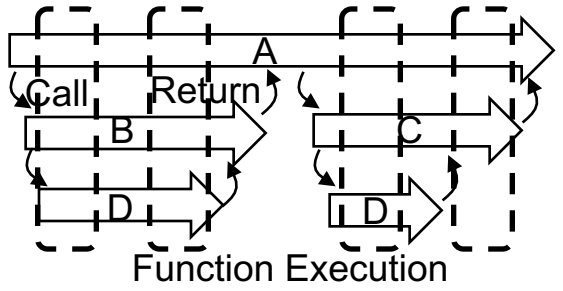
Function lifetime
A stack trace event
Conservative estimation

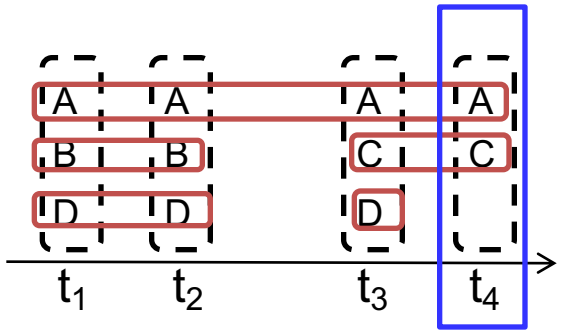
Captured Function Instances

B (T1-T2)
D (T1-T2)

Inference of Function Latencies

- Inference based on the continuity of a function in the context
  - Algorithm captures a period of a function execution in the call stack without a disruption of its context





Function lifetime
A stack trace event
Conservative estimation

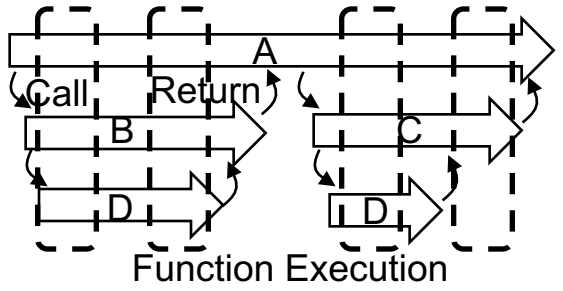
IsNew	ThisStack	Register (Time)
No	A	A (T1-**T4**)
No	C	C (**T3-T4**)

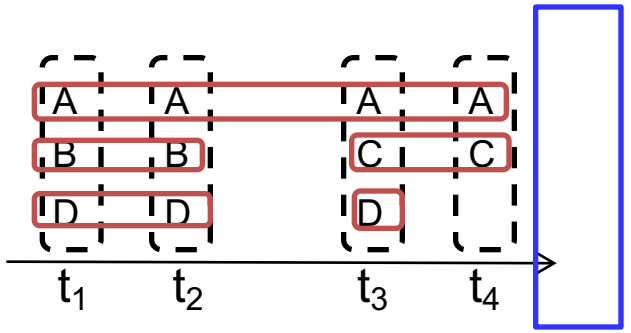
Captured Function Instances

B (T1-T2)
D (T1-T2)
D (T3-T3)

Inference of Function Latencies

- Inference based on the continuity of a function in the context
  - Algorithm captures a period of a function execution in the call stack without a disruption of its context





Function lifetime
A stack trace event
Conservative estimation

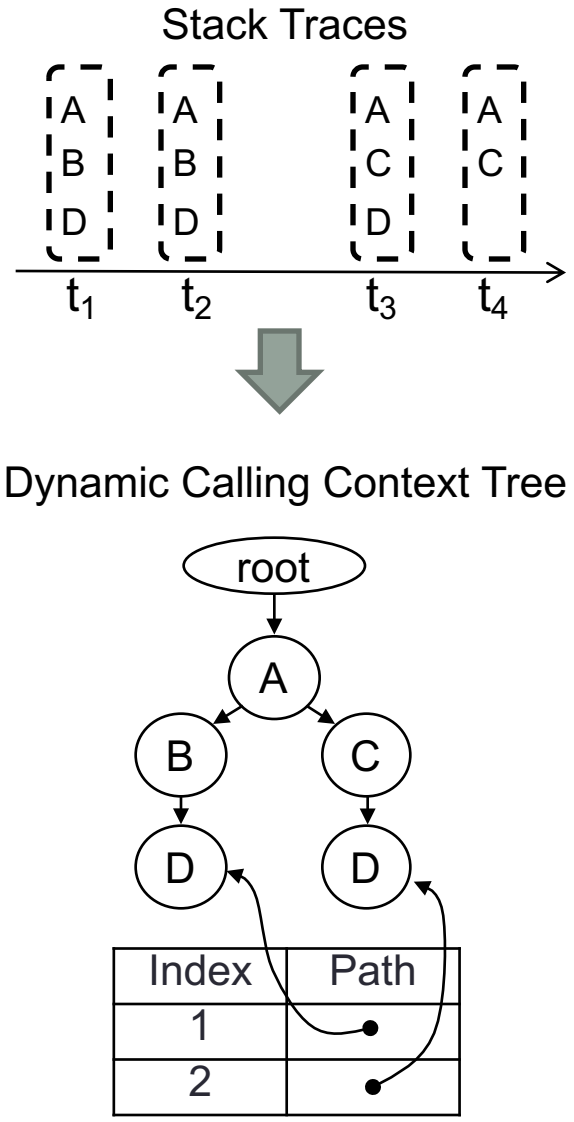
IsNew ThisStack Register (Time)

Captured Function Instances

B (T1-T2) D (T3-T3) **A (T1-T4)**
D (T1-T2) **C (T3-T4)**

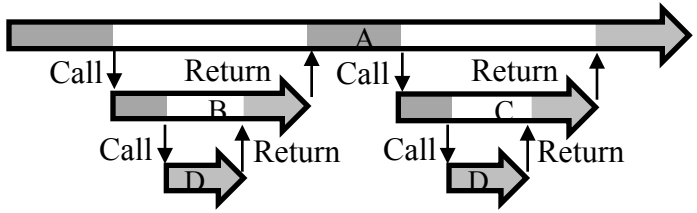
Dynamic Calling Context Tree

- A calling context is a distinct order of a function call sequence starting from the “main” function (i.e., a call path).
- We use calling context tree as the model of application performance to organize inferred latency in a structured way.
- Unique and concise index of a dynamic context is necessary for analysis.
  - Adopted a variant of the calling context tree data structure [Ammons97].
  - Assign a unique number of the pointer to the end of each path.



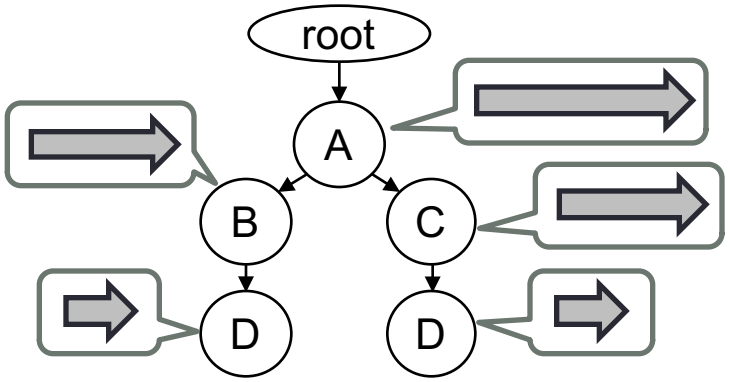
Performance-annotated Calling Context Tree

- Top-down Latency Normalization
  - Inference of latency performed in all layers of the stack causes overlaps of latencies in multiple layers.
  - Latency is normalized by recursively subtracting children functions' latencies in the calling context tree.

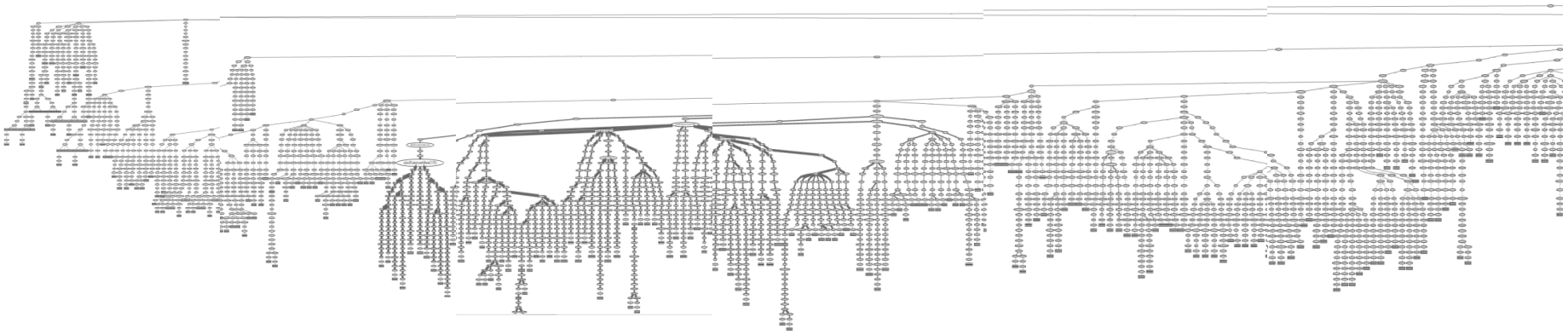


- Performance-annotated Calling Context Tree
  - Calling context tree is extended by annotating normalized inferred performance latencies in calling context tree.

Dynamic Calling Context Tree



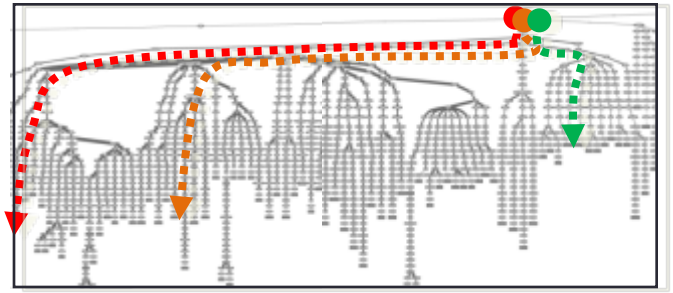
Context-sensitive Performance Analysis

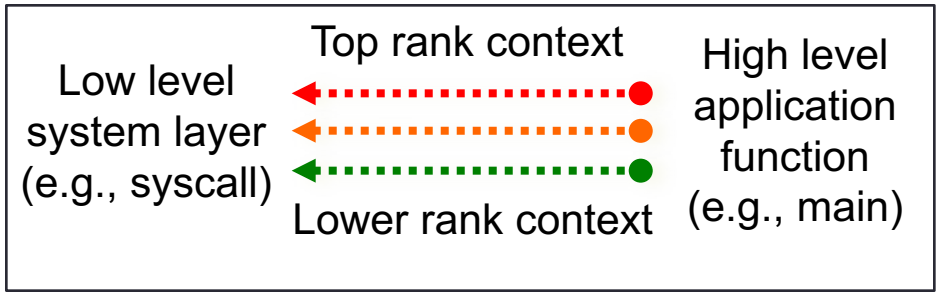


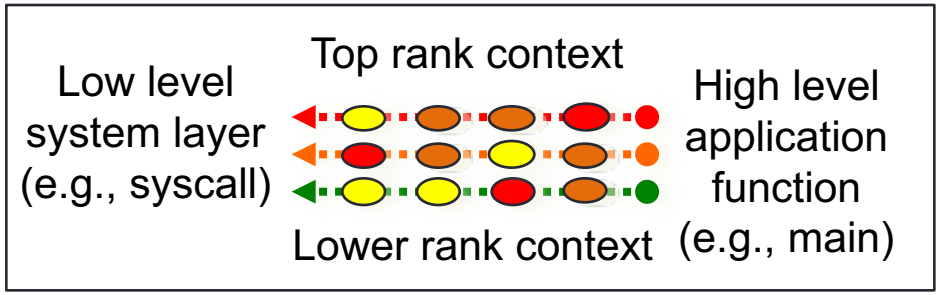
- Context-aware performance analysis involves diverse states of programs because of context-sensitive function call behavior.
- Manual analysis will consume significant time and efforts of users.
- Ranking of function call paths with latency allows us to focus on the sources of performance bug symptoms.

Ranking Calling Contexts and Functions

- We calculate the cost of each calling context (i.e., call path from the root) by storing the inferred function latencies.
- The top N ranked calling contexts regarding latency (i.e., hot calling contexts) are listed for evaluation.
- Furthermore, for each hot calling context, function nodes are ranked regarding their latencies and hot functions inside the path are determined.







Implementation

- IntroPerf is built on top of a production tracer, Event Tracing Framework for Windows (ETW).
- We use the stack traces generated on system calls and context switch events.
- Parser of ETW events and performance analyzer
  - 42K lines of Windows code in Visual C++
- Experiment machine
  - Intel Core i5 3.40 GHz CPU
  - 8GB RAM
  - Windows Server 2008 R2

Evaluation

Q1: How effective is IntroPerf at diagnosing performance bugs?

Q2: What is the coverage of program execution captured by system stack traces?

Q3: What is the runtime overhead of IntroPerf?

Evaluation – Performance Bugs

- **Q1: How effective is IntroPerf at diagnosing performance bugs?**

- Ranking of calling contexts and function instances allows developers to understand “where” and “how” performance bugs occur and determine the suitable code to be fixed.

- Evaluation Setup
  - Server programs (Apache, MySQL), desktop software (7zip), system utilities (ProcessHacker similar to the task manager)
  - Reproduced the cases of performance bugs. The ground truth of root causes are the patched functions.
  - Bug injection cases. The root causes are the injected functions.

- Two criteria depending on the locations of the bugs
  - Internal bugs and external bugs

Evaluation – Performance Bugs

- Internal Bugs
  - Performance bugs inside the main binary

Program Name	Bug ID	Program Characteristics				IntroPerf Evaluation				Internal/ External	Ground Truth
		$\|I\|$	$\|P\|$	$\|F\|$	l	p_{min}	f_{min}	Root Cause Binary	Root Cause Function		
Apache	45464	29	319	712	40.96	1	36	`libapr-1.dll`, Internal Library	`apr_stat`	Internal	Patch
MySQL	15811	36	1051	1275	31.22	1	0	`mysql.exe`, Main Binary	`strlen`	Internal	Patch
MySQL	49491	13	144	368	33.71	3	5	`mysqld.exe`, Main Binary	`Item_func_sha::val_str`	Internal	Patch
ProcessHacker	3744	23	2704	1172	49.34	1	0	`ProcessHacker.exe`, Main Binary	`PhSearchMemoryString`	Internal	Patch
7zip	S1	28	1160	1182	72.21	11	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch
7zip	S2	33	1793	1496	59.61	3	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch
7zip	S3	22	656	819	58.78	1	15	`7zFM.exe`, Main Binary	`CPanel::Post_Refresh_StatusBar`	Internal	Patch
7zip	S4	30	1002	1274	55.95	2	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch

Evaluation – Performance Bugs

- Internal Bugs
  - Performance bugs inside the main binary

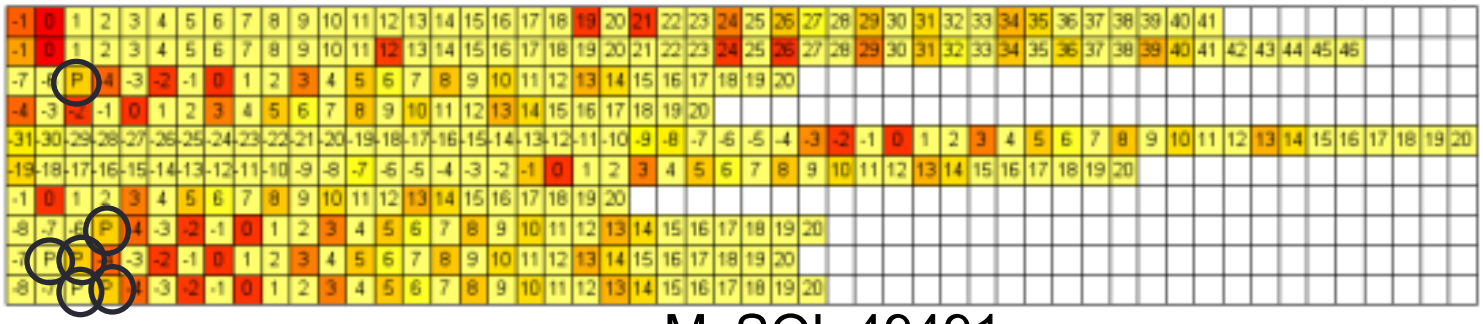
Program Name	Bug ID	Program Characteristics				IntroPerf Evaluation				Internal/ External	Ground Truth
		$\|I\|$	$\|P\|$	$\|F\|$	l	p_{min}	f_{min}	Root Cause Binary	Root Cause Function		
Apache	45464	29	319	712	40.96	1	36	`libapr-1.dll`, Internal Library	`apr_stat`	Internal	Patch
MySQL	15811	36	1051	1275	31.22	1	0	`mysql.exe`, Main Binary	`strlen`	Internal	Patch
MySQL	49491	13	144	368	33.71	3	5	`mysqld.exe`, Main Binary	`Item_func_sha::val_str`	Internal	Patch
ProcessHacker	3744	23	2704	1172	49.34	1	0	`ProcessHacker.exe`, Main Binary	`PhSearchMemoryString`	Internal	Patch
7zip	S1	28	1160	1182	72.21	11	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch
7zip	S2	33	1793	1496	59.61	3	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch
7zip	S3	22	656	819	58.78	1	15	`7zFM.exe`, Main Binary	`CPanel::Post_Refresh_StatusBar`	Internal	Patch
7zip	S4	30	1002	1274	55.95	2	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch

Low level system layer (e.g., system call) ⟷ High level application function (e.g., main)

Top rank context

↕

Lower rank context

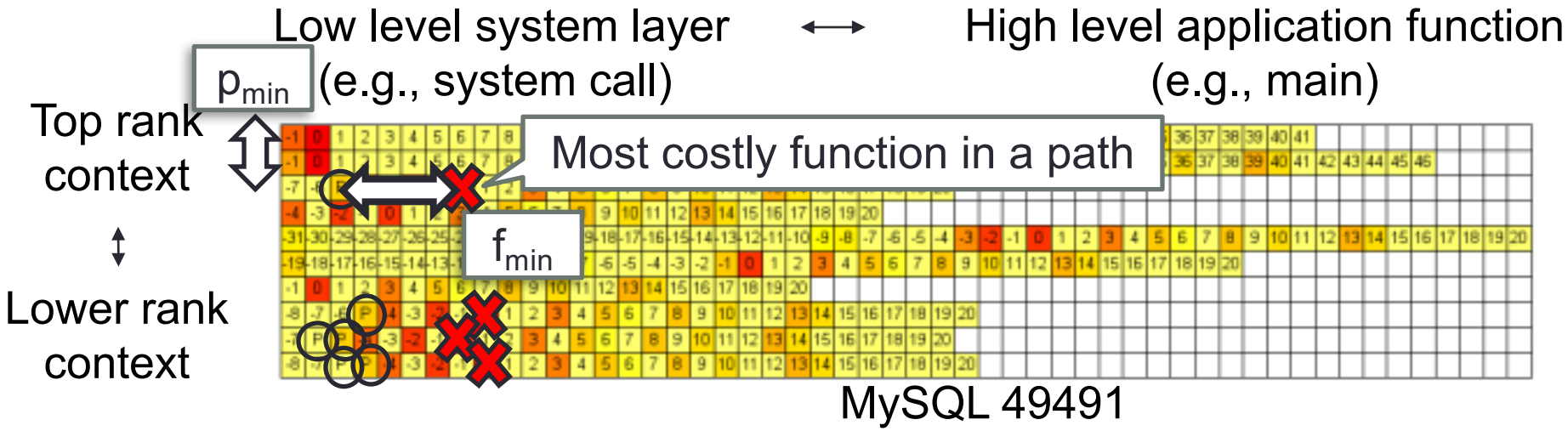


MySQL 49491

Evaluation – Performance Bugs

- Internal Bugs
  - Performance bugs inside the main binary

Program Name	Bug ID	Program Characteristics |I|	|P|	|F|	l	IntroPerf Evaluation p_{min}	f_{min}	Root Cause Binary	Root Cause Function	Internal/ External	Ground Truth
Apache	45464	29	319	712	40.96	1	36	libapr-1.dll, Internal Library	apr_stat	Internal	Patch
MySQL	15811	36	1051	1275	31.22	1	0	mysql.exe, Main Binary	strlen	Internal	Patch
MySQL	49491	13	144	368	33.71	3	5	mysqld.exe, Main Binary	Item_func_sha::val_str	Internal	Patch
ProcessHacker	3744	23	2704	1172	49.34	1	0	ProcessHacker.exe, Main Binary	PhSearchMemoryString	Internal	Patch
7zip	S1	28	1160	1182	72.21	11	16	7zFM.exe, Main Binary	CPanel::RefreshListCtrl	Internal	Patch
7zip	S2	33	1793	1496	59.61	3	16	7zFM.exe, Main Binary	CPanel::RefreshListCtrl	Internal	Patch
7zip	S3	22	656	819	58.78	1	15	7zFM.exe, Main Binary	CPanel::Post_Refresh_StatusBar	Internal	Patch
7zip	S4	30	1002	1274	55.95	2	16	7zFM.exe, Main Binary	CPanel::RefreshListCtrl	Internal	Patch



Evaluation – Performance Bugs

- Internal Bugs
  - Performance bugs inside the main binary

Program Name	Bug ID	Program Characteristics				IntroPerf Evaluation				Internal/ External	Ground Truth
		$\|I\|$	$\|P\|$	$\|F\|$	l	p_{min}	f_{min}	Root Cause Binary	Root Cause Function		
Apache	45464	29	319	712	40.96	1	36	`libapr-1.dll`, Internal Library	`apr_stat`	Internal	Patch
MySQL	15811	36	1051	1275	31.22	1	0	`mysql.exe`, Main Binary	`strlen`	Internal	Patch
MySQL	49491	13	144	368	33.71	3	5	`mysqld.exe`, Main Binary	`Item_func_sha::val_str`	Internal	Patch
ProcessHacker	3744	23	2704	1172	49.34	1	0	`ProcessHacker.exe`, Main Binary	`PhSearchMemoryString`	Internal	Patch
7zip	S1	28	1160	1182	72.21	11	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch
7zip	S2	33	1793	1496	59.61	3	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch
7zip	S3	22	656	819	58.78	1	15	`7zFM.exe`, Main Binary	`CPanel::Post_Refresh_StatusBar`	Internal	Patch
7zip	S4	30	1002	1274	55.95	2	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch

- External Bugs
  - Performance bugs outside the main binary

Program Name	Bug ID	Program Characteristics				IntroPerf Evaluation				Internal/ External	Ground Truth
		$\|I\|$	$\|P\|$	$\|F\|$	l	p_{min}	f_{min}	Root Cause Binary	Root Cause Function		
ProcessHacker	5424	25	1488	978	40.60	1	54	`ToolStatus.dll`, Plug-in	`MainWndSubclassProc`	External	Patch
ProcessHacker	-	26	1241	906	41.56	1	0	`ToolStatus.dll`, Plug-in	`NcAreaWndSubclassProc`	External	Inject
Internet Explorer	-	92	18716	6168	71.64	1	0	`MotleyFool.dll`, Toolbar Plug-in	`CMFToolbar::GetQuote`	External	Inject
Miranda	-	42	1032	1245	52.40	1	0	`Yahoo.dll`, Plug-in	`upload_file`	External	Inject
Apache	-	14	77	302	26.12	3	0	`mod_log_config.so`, Plug-in	`ap_default_log_writer`	External	Inject
Apache	-	18	96	331	25.89	2	0	`mod_deflate.so`, Plug-in	`deflate_out_filter`	External	Inject
VirtualBox	-	39	1288	1031	39.36	1	0	`QtCore4.dll`, External Library	`QEventDispatcherWin32::processEvents`	External	Inject

Evaluation – Performance Bugs

- Internal Bugs
  - Performance bugs inside the main binary

Program Name	Bug ID	Program Characteristics				IntroPerf Evaluation				Internal/ External	Ground Truth
		$\|I\|$	$\|P\|$	$\|F\|$	l	p_{min}	f_{min}	Root Cause Binary	Root Cause Function		
Apache	45464	29	319	712	40.96	1	36	`libapr-1.dll`, Internal Library	`apr_stat`	Internal	Patch
MySQL	15811	36	1051	1275	31.22	1	0	`mysql.exe`, Main Binary	`strlen`	Internal	Patch
MySQL	49491	13	144	368	33.71	3	5	`mysqld.exe`, Main Binary	`Item_func_sha::val_str`	Internal	Patch
ProcessHacker	3744	23	2704	1172	49.34	1	0	`ProcessHacker.exe`, Main Binary	`PhSearchMemoryString`	Internal	Patch
7zip	S1	28	1160	1182	72.21	11	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch
7zip	S2	33	1793	1496	59.61	3	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch
7zip	S3	22	656	819	58.78	1	15	`7zFM.exe`, Main Binary	`CPanel::Post_Refresh_StatusBar`	Internal	Patch
7zip	S4	30	1002	1274	55.95	2	16	`7zFM.exe`, Main Binary	`CPanel::RefreshListCtrl`	Internal	Patch

- External Bugs
  - Performance bugs outside the main binary

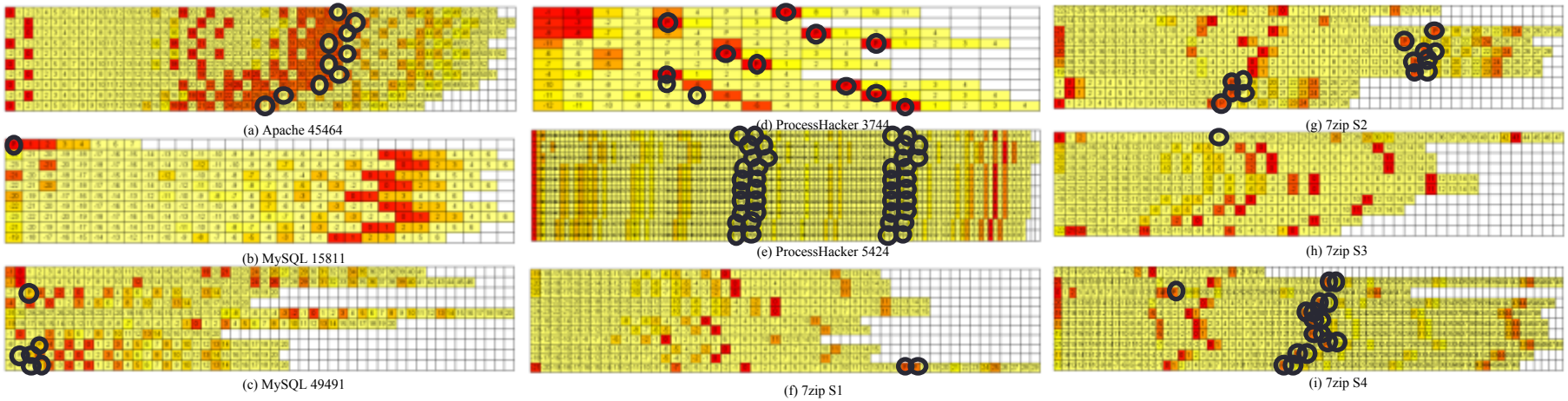
Program Name	Bug ID	Program Characteristics				IntroPerf Evaluation				Internal/ External	Ground Truth
		$\|I\|$	$\|P\|$	$\|F\|$	l	p_{min}	f_{min}	Root Cause Binary	Root Cause Function		
ProcessHacker	5424	25	1488	978	40.60	1	54	`ToolStatus.dll`, Plug-in	`MainWndSubclassProc`	External	Patch
ProcessHacker	-	26	1241	906	41.56	1	0	`ToolStatus.dll`, Plug-in	`NcAreaWndSubclassProc`	External	Inject
Internet Explorer	-	92	18716	6168	71.64	1	0	`MotleyFool.dll`, Toolbar Plug-in	`CMFToolbar::GetQuote`	External	Inject
Miranda	-	42	1032	1245	52.40	1	0	`Yahoo.dll`, Plug-in	`upload_file`	External	Inject
Apache	-	14	77	302	26.12	3	0	`mod_log_config.so`, Plug-in	`ap_default_log_writer`	External	Inject
Apache	-	18	96	331	25.89	2	0	`mod_deflate.so`, Plug-in	`deflate_out_filter`	External	Inject
VirtualBox	-	39	1288	1031	39.36	1	0	`QtCore4.dll`, External Library	`QEventDispatcherWin32::processEvents`	External	Inject

Evaluation – Performance Bugs

- **Summary : The root causes of all our evaluation cases are caught in the top 11 costly calling contexts.**

- The distance between costly functions and the patched functions differs depending on the types of bugs and application semantics.

- IntroPerf assists the patching process by presenting top ranked costly calling contexts and functions.



Evaluation – Coverage

Q2: What is the coverage of program execution captured by system stack traces?

- We measured how much dynamic program state is covered by stack traces in two criteria: dynamic calling contexts, function call instances
- We used a dynamic program instrumentation tool, Pin, to track all function calls, returns, and system calls and obtain the ground truth.
- Context switch events are simulated based on a reference to scheduling policies of Windows systems [Buchanan97].
- Three configurations are used for evaluation.
  1. System calls
  2. System calls with a low rate context switch events (120ms)
  3. System calls with a high rate context switch events (20ms)

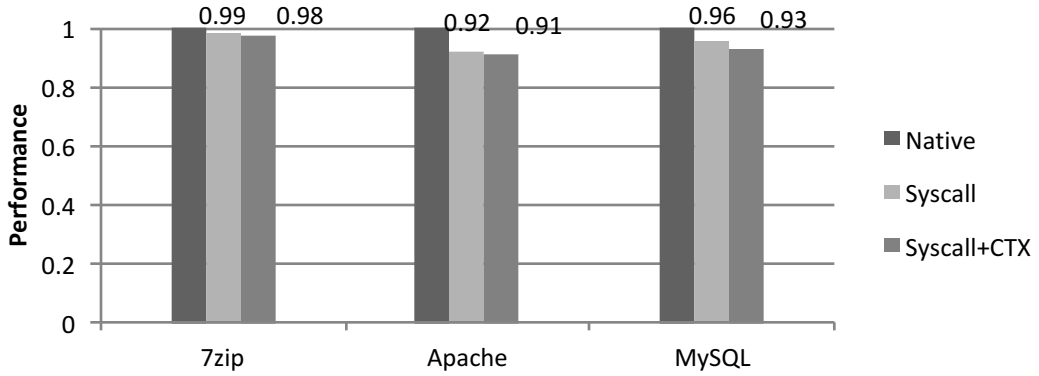
Evaluation – Coverage

- Coverage analysis of three applications: Apache, MySQL, and 7zip
  - System call rate: 0.33~2.78% for Apache, 0.21~1.48% for MySQL, 0.11~5.03% for 7zip
- Coverage for all:
  - Calling contexts: 5.3~49.4%
  - Function instances: 0.6~31.2%
- Coverage for **top 1% slowest functions**:
  - Calling contexts : **34.7~100%**
  - Function instances : **16.6~100%**
- **Summary: There is a significantly high chance to capture high latency functions which are important for performance diagnosis.**

Evaluation - Performance

Q3: What is the runtime overhead of IntroPerf?

- Evaluation of Windows ETW's performance for generating stack traces of three applications: Apache, MySQL, and 7zip
- Tracing overhead
  - Stack traces on system calls: **1.37~8.2%**
  - Stack traces on system calls and context switch events: **2.4~9.11%**
- Reasonable to be used in a post-development stage



Conclusion

- IntroPerf provides a transparent performance introspection technique based on the inference of function latencies from system stack traces.
- We evaluated IntroPerf on a set of widely used open source software and automatically found the root causes of real world performance bugs and delay-injected cases.
- The results show the effectiveness and practicality of IntroPerf as a lightweight performance diagnosis tool in a post-development stage.

Thank you


NEC Laboratories
America
Relentless passion for innovation


PURDUE
UNIVERSITY