The Efficient Server Audit Problem, Deduplicated Re-execution, and the Web

Cheng Tan, Lingfan Yu, Joshua B. Leners*, and Michael Walfish

NYU Department of Computer Science, Courant Institute
*Two Sigma Investments
Amazon Web Services

company

Alice

employee

employee

Amazon Web Services

- wiki PHP
- PHP runtime
- web server
- OS
- hypervisor
- hardware

database

request

response
• Alice has confidence in the wiki's PHP code
• Alice has confidence in the wiki's PHP code
• Still, lots of things can go wrong ...
• Alice has confidence in the wiki's PHP code
• Still, lots of things can go wrong ...
• Alice has confidence in the wiki's PHP code
• Still, lots of things can go wrong ...
• Alice has confidence in the wiki's PHP code
• Still, lots of things can go wrong ...
- Alice has confidence in the wiki's PHP code
- Still, lots of things can go wrong ...
Alice has confidence in the wiki's PHP code
Still, lots of things can go wrong ...
• Alice has confidence in the wiki's PHP code
• Still, lots of things can go wrong ...
• Alice has confidence in the wiki's PHP code
• Still, lots of things can go wrong ...
• Thus, Alice wants to **audit** the delivered responses
  – Are they derived from executing the **actual** application?
The Efficient Server Audit Problem
The Efficient Server Audit Problem

online phase

server

requests

clients

responses

program
The Efficient Server Audit Problem

1. server is untrusted; can respond arbitrarily
2. server is concurrent
The Efficient Server Audit Problem

1. server is untrusted; can respond arbitrarily
2. server is concurrent
The Efficient Server Audit Problem

1. server is untrusted; can respond arbitrarily
2. server is concurrent
The Efficient Server Audit Problem

1. server is untrusted; can respond arbitrarily
2. server is concurrent
The Efficient Server Audit Problem

1. server is untrusted; can respond arbitrarily
2. server is concurrent
3. verifier is weaker than server
4. server overhead is low; legacy applications supported
The Efficient Server Audit Problem

1. server is untrusted…
2. server is concurrent
3. verifier is weaker than server
4. server overhead is low...

The diagram shows the online phase and audit phase of the server audit problem. The clients communicate with the server through the requests and responses, with the trace collector and server forming the online phase. The audit phase involves the verifier examining the requests, responses, and program, concluding with

\[ \text{requests} + \text{program} \equiv \text{responses} \]
The Efficient Server Audit Problem

- Combination of these four is a new problem.
- Execution integrity is complementary to program verification.
What about naive re-execution?

1. server is untrusted…
2. server is concurrent
3. verifier is weaker than server
4. server overhead is low...
What about naive re-execution?

1. server is untrusted…
2. server is concurrent
3. verifier is weaker than server
4. server overhead is low...

1. server is untrusted…
2. server is concurrent
3. verifier is weaker than server
4. server overhead is low…
What about naive re-execution?

- This does not save the verifier work.
What about naive re-execution?

- This does not save the verifier work.
- Instead, we will accelerate re-execution.
Rest of the talk

1. How does the verifier accelerate re-execution? (these two are in tension)

2. Why are shared objects (such as DBs) challenging?

3. Does our implementation for PHP perform well?
Rest of the talk

1. How does the verifier accelerate re-execution?

2. Why are shared objects (such as DBs) challenging?

3. Does our implementation for PHP perform well?
Accelerating re-execution: a 30,000-foot view

- Deduplicate computation across requests
Poirot’s observation: repeated computation

Poirot’s observation: repeated computation

T. Kim, R. Chandra, and N. Zeldovich.
Efficient patch-based auditing for web applications. *OSDI*, 2012
Poirot’s observation: repeated computation

Poirot’s observation: repeated computation

requires trusting the advice

T. Kim, R. Chandra, and N. Zeldovich.
Efficient patch-based auditing for web applications. *OSDI*, 2012
We accelerate re-execution without trusting the server

server (online)

\[ C: \text{tag} \rightarrow \{\text{set of reqs}\} \]

for each tag:
- execute \( C(\text{tag}) \) with SIMD-on-demand
- conduct unanimity checks

verifier (offline)
We accelerate re-execution without trusting the server

For each tag:
- execute $C(\text{tag})$ with SIMD-on-demand
- conduct unanimity checks

SIMD-on-demand re-executes identical instructions once.
SIMD-on-demand eliminates redundant computation

main(a,b):
c ← a * b

req_i: a=1; b=2
req_j: a=2; b=1
SIMD-on-demand eliminates redundant computation

main(a,b):
  c ← a * b
  c ← c + 1

req\textsubscript{i}:
\begin{align*}
a &= 1; b &= 2 \\
\end{align*}

req\textsubscript{j}:
\begin{align*}
a &= 2; b &= 1 \\
\end{align*}

- Multi-value represents different values of the same variable.
SIMD-on-demand eliminates redundant computation

\[
\text{main}(a,b): \\
\quad c \leftarrow a \times b \\
\quad c \leftarrow c + 1
\]

\[c = 3\]

\[a = [1,2] \quad b = [2,1]\]

\[c = 2\]

\[+1 \quad c = 3\]

\[\text{req}_i: \quad a = 1; b = 2\]

\[\text{req}_j: \quad a = 2; b = 1\]

- **Multi-value** represents different values of the same variable.
- **Verifier** **collapses** multi-value to scalar if possible.
Recap

- Verifier re-executes in an accelerated way …

- ... by exploiting advice from the server ...

- ... without trusting that advice.
1. How does the verifier accelerate re-execution?

2. Why are shared objects (such as DBs) challenging?

3. Does our implementation for PHP perform well?
• Will try to give some intuition for the difficulties

• Solutions in the paper, rigorous proofs in tech report
• Will try to give some intuition for the difficulties

• Solutions in the paper, rigorous proofs in tech report

• For now, assume simple storage model
  – Read-write registers, named with letters
Central challenge: re-execution is out of order

server (online)

- `write(A, 56)`
- `56 ← read(A)`
- `write(B, 12)`
- `12 ← read(B)`

register A

register B

verifier (offline)

register B
Central challenge: re-execution is out of order

server (online)

write(A, 56)

56 ← read(A)

write(B, 12)

12 ← read(B)

register A

register B

verifier (offline)

...
Central challenge: re-execution is out of order

server (online)

write(A, 56)

56 ← read(A)

write(B, 12)

12 ← read(B)

register A

register B

server (online)

write(B, 12)

? ← read(B)

register B

verifier (offline)
How can the verifier re-execute reads? Attempt 1:

- Server logs read values; verifier supplies from log

```
write(A,56)  →   register A
56←read(A)
write(B,12)  →   register B
12←read(B)
```

```
server (online)

write(B,12)
12←read(B)
```

```
verifier (offline)

write(B,12)
12←read(B)
```
How can the verifier re-execute reads? Attempt 1:

- Server logs read values; verifier supplies from log
- This can fool the verifier
How can the verifier re-execute reads? Attempt 2:

- Server: logs operands
- Verifier: simulates reads using log and checks writes
How can the verifier re-execute reads? Attempt 2:

- Server: logs operands
- Verifier: simulates reads using log and checks writes
How can the verifier re-execute reads? Attempt 2:

Server: logs operands
Verifier: simulates reads using log and checks writes
Another challenge is validating the log order

• Order in log could be nonsensical

• Verifier must check consistency of log:
  – Is log order consistent with observed request order?

• This check must be efficient
1. How does the verifier accelerate re-execution?

2. Why are shared objects (such as DBs) challenging?

3. Does our implementation for PHP perform well?
A built system: Orochi

- Orochi targets apps based on PHP and SQL ("LAMP")
- Server and verifier: modified PHP runtimes
- Includes techniques for deduplicating database queries
- Details
  - Built atop HipHop VM (HHVM)
  - 20K lines of C++, PHP, Bash, Python
Evaluation questions

• Is auditing efficient for the verifier?

• What is the price of verifiability?

• How compatible is Orochi with legacy applications?
• **Applications:**
  – MediaWiki, phpBB and HotCRP

• **Workloads:**
  – phpBB: 7-day’s posts from CentOS forum
  – HotCRP: Simulation of SIGCOMM’09
Our workloads see a lot of redundant computation

MediaWiki’s workload (20K requests)
Orochi's verifier is efficient

Orochi's verifier achieves speedups compared to naive replay

* Pessimistically estimated from the original online execution
The price of verifiability is tolerable

<table>
<thead>
<tr>
<th>CPU</th>
<th>Network</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediaWiki’s workload</td>
<td>trace (per req)</td>
<td>MediaWiki’s workload</td>
</tr>
<tr>
<td>4.7%</td>
<td>advice (per req)</td>
<td>1.0x</td>
</tr>
<tr>
<td>phpBB’s workload</td>
<td>7.1KB</td>
<td>phpBB’s workload</td>
</tr>
<tr>
<td>8.6%</td>
<td>Orochi’s overhead</td>
<td>1.7x</td>
</tr>
<tr>
<td>HotCRP’s workload</td>
<td>5.7KB</td>
<td>HotCRP’s workload</td>
</tr>
<tr>
<td>5.9%</td>
<td></td>
<td>1.5x</td>
</tr>
</tbody>
</table>

Verifier needs to store the trace and advice for one audit epoch.
Orochi requires modest application adjustments

- Lines of code modified:
  - 346 lines of code change for MediaWiki
  - 270 lines of code change for phpBB
  - 67 lines of code change for HotCRP

- Most of the changes are due to
  - PHP features that our implementation does not support
  - Modifying the application to respect object semantics
Recap of evaluation

- Verifier: 5.6--10.9x speedup over naive re-execution
- Costs: storage at verifier, <10% overhead on server
- Compatibility: Modest application changes
Related work, future work, and wrap-up
Related work

• Efficient execution integrity
  – Replication: BFT
  – Attestation: TPMs, SGX
  – Probabilistic proofs: Pepper, CMT, Pinocchio, Pantry, SNARKs

• Computation deduplication (Delta execution, iThreads)

• Record-replay
  – Untrusted recorder: Accountable Virtual Machines
  – Accelerated replayer: Poirot
  – Multiprocessor: RecPlay, LEAP, DoublePlay, PRES, ODR, …
Related work

• Efficient execution integrity
  – Replication: BFT
  – Attestation: TPMs, SGX
  – Probabilistic proofs: Pepper, CMT, Pinocchio, Pantry, SNARKs

• Computation deduplication (Delta execution, iThreads)

• Record-replay
  – Untrusted recorder: Accountable Virtual Machines
  – Accelerated replayer: Poirot
  – Multiprocessor: RecPlay, LEAP, DoublePlay, PRES, ODR, …
Related work

- **Efficient execution integrity**
  - Replication: BFT
  - Attestation: TPMs, SGX
  - Probabilistic proofs: Pepper, CMT, Pinocchio, Pantry, SNARKs

- **Computation deduplication (Delta execution, iThreads)**

- **Record-replay**
  - Untrusted recorder: Accountable Virtual Machines
  - Accelerated replayer: Poirot
  - Multiprocessor: RecPlay, LEAP, DoublePlay, PRES, ODR, …
Wrap-up and future work

• Our solution to the Efficient Server Audit Problem:
  – Includes a new accelerated re-execution technique
  – Includes new algorithms for verifying concurrent executions
  – Comes with a rigorous proof of correctness

• Our instantiation for PHP, SQL web apps:
  – 5-10x speedups over a naive replay; <10% CPU overhead on server

• Future work includes:
  – SGX integration
  – Extend to multiple interacting servers
  – Accelerate other record-replay systems