Spice: Verifiable state machines
A foundation for building high-throughput confidential blockchains

Srinath Setty, Sebastian Angel, Trinabh Gupta, and Jonathan Lee
Microsoft Research                         UT Austin and NYU
Verifiable state machines

Clients

Verifiers

Accept/reject

Tests

Concurrent requests

Responses

Append-only trace

Stateful service

Store

Fully untrusted (no trusted hardware)
Verifiable state machines

- **Correctness:** If the service’s behavior is equivalent to a serial execution, verifiers accept.
- **Soundness:** If the service misbehaves, Pr[ a verifier outputs accept ] < $\epsilon$
- **Zero-knowledge:** Trace does not reveal *anything* beyond correct execution
- **Succinctness:** Each entry in the trace is small
Prior work on verifiable state machines

• The underlying theory dates back to 90s: Babai et al. [STOC91]

Cost reductions by $10^{20}$x

• Pepper [HotOS11, NDSS12], CMT [ITCS12], Ginger [Security12], TRMP [HotCloud12]

Support stateful computations

• Zaatar [EuroSys13], Pinocchio [Oakland13], SNARKS-for-C [CRYPTO13]

• Pantry [SOSP13], Geppetto [Oakland15], CTV [EUROCRYPT15], vSQL [Oakland17], ...

Storage interfaces: key-value stores, SQL databases, etc.
Two key limitations of recent systems:
- Storage ops. are expensive: tens of seconds to minutes of CPU-time
- Support only a sequential execution model

Prior work can only support < 0.15 reqs/sec (even for simple services)

Storage interfaces: key-value stores, SQL databases, etc.
Two key limitations of recent systems:
• Storage ops. are expensive: tens of seconds to minutes of CPU-time
• Support only a sequential execution model

Prior work can only support < 0.15 reqs/sec (even for simple services)

Our system, Spice [OSDI18, to appear]:
• Features a storage primitive that is >100x faster
• Supports a concurrent execution model
• Throughput: 488—1048 reqs/sec (512 CPU-cores)
Rest of this talk

• Applications of verifiable state machines

• Background and overview of Spice

• Experimental results
Why are we interested in verifiable state machines?

They enable us to build:

1. Cloud services *without trusting* the cloud infrastructure

2. *Private and efficient* blockchains
   - Permissionless (e.g., Ethereum)
   - Permissioned (e.g., Hyperledger Fabric, Quorum, etc.)
Cloud-hosted ledgers
(inspired by https://sequence.io)

Value proposition:
An auditor can verify the service---without access to requests or trusting the service

1. Issue(clientId, asset, balance, issuerSig)
2. Transfer(senderId, recvId, asset, amount, senderSig)
3. Retire(clientId, asset, amount, clientSig)
Private and fast smart contracts

- All transactions and contract state are public → no confidentiality
- Every app-level request must be processed by blockchain → limits throughput

- Only a succinct trace is public → strong confidentiality
- Ethereum processes a succinct trace → can support high-throughput apps
• Applications

• **Background and overview of Spice**

• Experimental results
A quick overview of Pantry\textsuperscript{[SOSP13]}

• Extends Zaatar\textsuperscript{[EuroSys13]} and Pinocchio\textsuperscript{[Oakland13]} to support state

\begin{align*}
\text{int } f(\text{int } a, \text{int } b) \{ \\
\quad \text{return } a*2 - b; \\
\}
\end{align*}

C program

\begin{align*}
\text{x}_1 - \text{input } a &= 0 \\
\text{x}_2 - \text{input } b &= 0 \\
(\text{x}_1 \cdot 2) - \text{x}_3 &= 0 \\
(\text{x}_3 - \text{x}_2) - \text{x}_4 &= 0 \\
\text{x}_4 - \text{output} &= 0
\end{align*}

Equations over variables in a large (256-bit) finite field $\mathbb{F}_p$

\begin{align*}
\text{verifier} &
\quad \text{build}\quad a = 3 \\
\quad \text{y = 2 proof}
\end{align*}

\begin{align*}
\text{prover} &
\quad \text{translate}\quad \text{int } f(\text{int } a, \text{int } b) \{ \\
\quad \quad \text{return } a*2 - b; \\
\quad \}
\end{align*}
Zaatar and Pinocchio support a large C subset:

- Arithmetic operations, bitwise operations
- Conditional control flow
- Volatile memory including pointers
- Loops: bound must be known at compile time

Pantry supports state while working in a stateless model---by using cryptographic hashes (a folklore idea)
How does Pantry support state?

Key idea: name data blocks with their short cryptographic hashes

```
int req(int a, uint8_t *digest) {
  int b = prover_input();
  assert(digest == SHA256(b));
  return a * 2 - b;
}
```

If prover supplies invalid state the assert will fail

Cost of a key-value store operation: logarithmic in the size of state
Concretely: several minutes of CPU-time (10^6 KV pairs)

Pantry builds a key-value store using this idea: treat hashes as pointers to data and construct a (Merkle) tree
Spice in a nutshell

Core idea: Use a **set data structure** [Blum et al. FOCS91, Clarke et al. ASIACRYPT03, Arasu et al. SIGMOD17] instead of a tree

- Key-value store op = add an element to a set
- Costs = constant-time (amortized) vs. logarithmic
Spice in a nutshell

Core idea: Use a set data structure [Blum et al. FOCS91, Clarke et al. ASIACRYPT03, Arasu et al. SIGMOD17] instead of a tree

• Key-value store op = add an element to a set
• Costs = constant-time (amortized) vs. logarithmic

However: A naïve instantiation is slower than tree-based approach

• Spice presents an efficient instantiation using ECC (10^6x faster than naïve)
• Spice includes new techniques to support inexpensive transactions
• ....
Implementation of Spice

- 3,000 lines of C atop Pantry [SOSP13] (~15,000 LOC)
- Three apps: 1,500 lines of C

- `request handler` (a C program with calls to Spice’s storage API)
  - `init()`, `insert(Key, Value)`, `put(Key, Value)`, `get(Key)`, `delete(Key)`
  - `lock(Key)`, `unlock(Key)`
  - `begin_txn(Key[], Value**)`, `end_txn(Key[], Value[])`
• Applications

• Background and overview of Spice

• Experimental results
Evaluation questions

1. How does Spice compare with the prior state-of-the-art?
2. What is the end-to-end performance of apps built with Spice?

Evaluation testbed:

- Azure D64s_v3 instances: 64 vCPUs, 2.4Ghz Intel Xeon, 256 GB RAM, running Ubuntu 17.04
How does Spice compare to prior work?

A million key-value pairs

Transactions with a single operation, keys chosen with a uniform distribution

Metric: number of operations/second

<table>
<thead>
<tr>
<th></th>
<th>get</th>
<th>put</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pantry [SOSP13]</td>
<td>0.078</td>
<td>0.039</td>
</tr>
<tr>
<td>Pantry++</td>
<td>0.15</td>
<td>0.076</td>
</tr>
<tr>
<td>Geppetto [Oakland15]</td>
<td>0.002</td>
<td>0.002</td>
</tr>
<tr>
<td>Spice (1-thread)</td>
<td>3.3</td>
<td>3.3</td>
</tr>
<tr>
<td>Spice (512-threads)</td>
<td>1,250</td>
<td>1,259</td>
</tr>
</tbody>
</table>
(2) End-to-end performance with varying #CPUs

- TPS is 16,000x better than prior state-of-the art (algorithmic + hardware)
- Verification throughput: 15 million proof verifications/second
Summary

• Verifiable state machines is a key tool for the cloud and blockchains

• Spice is a substantial milestone for building verifiable state machines
  • >16,000x better performance (over prior state-of-the-art)
  • Supports real-world applications with thousands of transactions/sec

• We are excited about the many possibilities Spice points to!
Thank you!