Shortstack: Distributed, Fault-tolerant, Oblivious Data Access

Midhul
Kushal
Anurag
Rachit

Cornell University
Yale University

(First two authors contributed equally)
Oblivious Data Access
Oblivious Data Access

Trusted

Client

Client

Untrusted

Key-Value Store

- 41b
- 7f5
- c16
- 2ce
Oblivious Data Access

Trusted

Client

Client

Untrusted

Key-Value Store

41b

7f5

c16

2ce
Oblivious Data Access

Truste\(d\)

Client

Client

Untrusted

Key-Value Store

Output
Access distribution

41b 7f5 c16 2ce

41b 7f5 41b 7f5 41b

Oblivious Data Access

Client

Trusted

Input Access distribution

a b c d

Untrusted Key-Value Store

Key-Value Store

41b
7f5
c16
2ce

Output Access distribution

41b 7f5 c16 2ce

a
Oblivious Data Access

Encryption is not enough. Need to hide access patterns
Oblivious Data Access

Input Access distribution

Output Access distribution

Encryption is not enough. Need to hide access patterns

Oblivious Data Access: Output distribution independent of input distribution
Existing Oblivious Data Access Techniques

Untrusted Key-Value Store

| 41b | 7f5 | c16 | 2ce |

Trusted

Client

Client

Input Distribution

a b c d

Output Distribution

41b 7f5 c16 2ce
Existing Oblivious Data Access Techniques

Input Distribution

Output Distribution

Key-Value Store

Untrusted

Proxy

Client

Client

Trusted

41b
7f5
c16
2ce
Existing Oblivious Data Access Techniques

Existing techniques “flatten” input access distribution into uniform output distribution
Existing Oblivious Data Access Techniques

Existing techniques “flatten” input access distribution into **uniform** output distribution

**ORAM**

- **Bandwidth Overhead**: Large $O(\log n)$
- **Adversarial Model**: Active
  (Can inject queries)
Existing Oblivious Data Access Techniques

Existing techniques “flatten” input access distribution into uniform output distribution.

**Input Distribution**
- a
- b
- c
- d

**Output Distribution**
- 41b
- 7f5
- c16
- 2ce

**ORAM**
- Bandwidth Overhead: Large $O(\log n)$
- Adversarial Model: Active (Can inject queries)

**Pancake**
- Bandwidth Overhead: Constant (3x)
- Adversarial Model: Honest-but-curious (Does not inject queries)
Existing Oblivious Data Access Techniques

Existing techniques “flatten” input access distribution into uniform output distribution.

All existing oblivious data access techniques use centralized (stateful) proxy.
This work
This work

Challenges with Centralized Oblivious Data Access Systems
1. Insecure or unavailable during failures
2. Scalability bottleneck
Challenges with Centralized Oblivious Data Access Systems

1. Insecure or unavailable during failures
2. Scalability bottleneck

Design of Shortstack:
Distributed, Fault-tolerant, Oblivious Data Access System
Challenges with Centralized Oblivious Data Access Systems
1. Insecure or unavailable during failures
2. Scalability bottleneck

Design of Shortstack:
Distributed, Fault-tolerant, Oblivious Data Access System

Security model:
Enables formal study of Distributed, Fault-tolerant, Oblivious Data Access systems
Challenges with Centralized Oblivious Data Access Systems
1. Insecure or unavailable during failures
2. Scalability bottleneck

Design of Shortstack:
Distributed, Fault-tolerant, Oblivious Data Access System

Security model:
Enables formal study of Distributed, Fault-tolerant, Oblivious Data Access systems
Pancake overview
Pancake overview

Three important functionalities:
• Replica Creation
• Query Generation
• Query Execution (+temporarily buffer writes to replicas)
Pancake overview

Three important functionalities:
- Replica Creation
- Query Generation
- Query Execution (+temporarily buffer writes to replicas)

Input Distribution

Keys: a, b, c, d
Pancake overview

Three important functionalities:
- **Replica Creation**
- **Query Generation**
- **Query Execution (+temporarily buffer writes to replicas)**

Three important functionalities:
- Replica Creation
- Query Generation
- Query Execution (+temporarily buffer writes to replicas)

![Diagram showing Pancake overview with Trusted and Untrusted components, and an Input Distribution chart with keys a, b, c, d and values 41b, 9ea]
Pancake overview

Three important functionalities:
• Replica Creation
• Query Generation
• Query Execution (+temporarily buffer writes to replicas)

Input Distribution

Keys

a b c d

Client

Client

Proxy

a ➔ 41b, 9ea
b ➔ 7f5

Trusted

Untrusted

Key-Value Store

41b
9ea
7f5
Three important functionalities:

- **Replica Creation**
- **Query Generation**
- **Query Execution (+temporarily buffer writes to replicas)**
Pancake overview

Three important functionalities:

- Replica Creation
- Query Generation
- Query Execution (+temporarily buffer writes to replicas)
Pancake overview

Three important functionalities:

- Replica Creation
- Query Generation
- Query Execution (+temporarily buffer writes to replicas)

Three important functionalities:

- Replica Creation
- Query Generation
- Query Execution (+temporarily buffer writes to replicas)
Pancake overview

Three important functionalities:
- Replica Creation
- Query Generation
- Query Execution (+temporarily buffer writes to replicas)

![Diagram showing Pancake overview]
Pancake overview

Three important functionalities:

- Replica Creation
- Query Generation
- Query Execution (+temporarily buffer writes to replicas)
Pancake overview

Three important functionalities:
- Replica Creation
- Query Generation
- Query Execution (+temporarily buffer writes to replicas)

Input Distribution

```
<table>
<thead>
<tr>
<th>Keys</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>41b, 9ea</td>
<td>7f5</td>
<td>c16</td>
<td>2ce</td>
</tr>
</tbody>
</table>
```

Output Distribution

```
| Replicas | 41b | 9ea | 7f5 | c16 | 2ce |
```

Proxy is stateful
Challenges with Centralized and Stateful Proxy
Challenges with Centralized and Stateful Proxy
Challenges with Centralized and Stateful Proxy

Write(a, b)
Challenges with Centralized and Stateful Proxy

- **Write** (a, □)

- **Proxy**

- **Trusted**

- **Untrusted**

- **Key-Value Store**

- **41b**

- **9ea**

- **7f5**

- **c16**

- **2ce**
Challenges with Centralized and Stateful Proxy

Proxy

Untrusted

Key-Value Store

41b
9ea
7f5
c16
2ce

Client

Write(a, )

Trusted

Client

a →
Challenges with Centralized and Stateful Proxy
Challenges with Centralized and Stateful Proxy
Challenges with Centralized and Stateful Proxy
Challenges with Centralized and Stateful Proxy

Inconsistency across replicas
Challenges with Centralized and Stateful Proxy

Inconsistency across replicas

- Fetch all the replicas of \( a \) to determine the latest value

Violates Security (correlated accesses)
Challenges with Centralized and Stateful Proxy

Inconsistency across replicas

- Fetch all the replicas of a to determine the latest value
- Violates Security (correlated accesses)
Challenges with Centralized and Stateful Proxy

- Inconsistency across replicas:
  - Fetch all the replicas of `a` to determine the latest value
  - Violates Security (correlated accesses)

- Security violations:
  - Fetch the entire database to avoid security violations
  - Huge bandwidth overhead (large periods of unavailability)
Challenges with Centralized and Stateful Proxy

- Inconsistency across replicas
  - Fetch all the replicas of \( a \) to determine the latest value
  - Violates Security (correlated accesses)

- Security violation or long periods of unavailability
  - Fetch the entire database to avoid security violations
  - Huge bandwidth overhead (large periods of unavailability)

Centralized, Stateful, Proxy:
Security violation or long periods of unavailability
This work

Challenges with Centralized Oblivious Data Access Systems
1. Insecure or unavailable during failures
2. Scalability bottleneck

Design of Shortstack:
Distributed, Fault-tolerant, Oblivious Data Access System

Security model:
Enables formal study of Distributed, Fault-tolerant, Oblivious Data Access systems
Shortstack Summary

Client

Trusted

Untrusted

Key-Value Store

41b
9ea
75f
c16
2ce
Shortstack Summary

- **Trusted**
  - Client
  - Proxy
  - Proxy

- **Untrusted**
  - Key-Value Store
    - 41b
    - 9ea
    - 75f
    - c16
    - 2ce
1. Oblivious data access guarantees, even under failures
   - Fail-stop failure model
   - Worst-case scenario: Arbitrary (bounded number of) failures at arbitrary times
1. Oblivious data access guarantees, even under failures
   - Fail-stop failure model
   - Worst-case scenario: Arbitrary (bounded number of) failures at arbitrary times

2. System Availability
1. Oblivious data access guarantees, even under failures
   • Fail-stop failure model
   • Worst-case scenario: Arbitrary (bounded number of) failures at arbitrary times

2. System Availability

3. Scalability: Alleviate bandwidth & compute bottlenecks, throughput linear in #physical-servers
1. Oblivious data access guarantees, even under failures
   - Fail-stop failure model
   - Worst-case scenario: Arbitrary (bounded number of) failures at arbitrary times

2. System Availability

3. Scalability: Alleviate bandwidth & compute bottlenecks, throughput linear in #physical-servers

Threat Model: Honest-but-curious Adversary (Or, Passive persistent adversary—The Pancake model)
Shortstack Key Insight
Obliviousness requires output distribution to be independent of input distribution
Obliviousness requires output distribution to be \textit{independent} of input distribution.
Uniform distribution is one but not the only way to achieve independence.
Obliviousness requires output distribution to be independent of input distribution.
Uniform distribution is one but not the only way to achieve independence.
Obliviousness requires output distribution to be independent of input distribution.
Uniform distribution is one but not the only way to achieve independence.
Obliviousness requires output distribution to be independent of input distribution.
Uniform distribution is one but not the only way to achieve independence.
Obliviousness requires output distribution to be independent of input distribution. Uniform distribution is one but not the only way to achieve independence.
Shortstack Design Principle #1

Partition query execution by Replicas
Shortstack Design Principle #1

Partition query execution by Replicas

Trusted

Client

Proxy

Proxy

Untrusted

Key-Value Store

41b
9ea
7f5
c16
2ce

Output Distribution

41b 9ea 7f5 c16 2ce
Shortstack Design Principle #1

Partition query execution by Replicas

Trusted

Proxy

41b 7f5 2ce

Proxy

9ea c16

Untrusted

Key-Value Store

41b

9ea

7f5

c16

2ce

Output Distribution

41b 9ea 7f5 c16 2ce
Shortstack Design Principle #1

Partition query execution by Replicas

Upon failure, output distribution is uniform over a random subset of replicas.
Output distribution is independent of input distribution (realizing our key insight).
Shortstack Design Principle #2

Key-Value Store

Client

Proxy

Client

Untrusted

Output Distribution

41b  9ea  7f5  c16  2ce
Shortstack Design Principle #2

Partition Proxy state by Keys

Proxy State
- **a** → 41b, 9ea
- **b** → 7f5
- **c** → c16
- **d** → 2ce

Trusted
- **Client**

Untrusted
- **Key-Value Store**
  - 41b
  - 9ea
  - 7f5
  - c16
  - 2ce

Output Distribution
- 41b
- 9ea
- 7f5
- c16
- 2ce
Shortstack Design Principle #2

Partition Proxy state by Keys

Example to provide intuition

Proxy State

<table>
<thead>
<tr>
<th>Key</th>
<th>Trusted Proxy</th>
<th>Untrusted Proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>41b, 9ea</td>
<td>41b</td>
</tr>
<tr>
<td>b</td>
<td>7f5</td>
<td>7f5</td>
</tr>
<tr>
<td>c</td>
<td>c16</td>
<td>c16</td>
</tr>
<tr>
<td>d</td>
<td>2ce</td>
<td>2ce</td>
</tr>
</tbody>
</table>

Key-Value Store

Untrusted

Output Distribution
Shortstack Design Principle #2

Partition Proxy state by Keys

Example to provide intuition
Shortstack Design Principle #2

Partition Proxy state by Keys

Example to provide intuition

Trusted

Client

Untrusted

Key-Value Store

Proxy

<table>
<thead>
<tr>
<th>41b</th>
<th>7f5</th>
<th>2ce</th>
</tr>
</thead>
<tbody>
<tr>
<td>a → 41b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b → 7f5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>d → 2ce</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Proxy

<table>
<thead>
<tr>
<th>9ea</th>
<th>c16</th>
</tr>
</thead>
<tbody>
<tr>
<td>a → 9ea</td>
<td></td>
</tr>
<tr>
<td>c → c16</td>
<td></td>
</tr>
</tbody>
</table>

41b
9ea
7f5
c16
2ce
Shortstack Design Principle #2

Partition Proxy state by Keys

Example to provide intuition

Would need synchronization
Expensive!

Trusted

Client

Client

Untrusted

Key-Value Store

Proxy

41b  7f5  2ce

a → 41b
b → 7f5
d → 2ce

Proxy

9ea  c16

a → 9ea
c → c16

41b
9ea
7f5
c16
2ce
Shortstack Design Principle #2

Partition Proxy state by Keys

Trusted

Client

Proxy

a ➔ 41b 9ea
b ➔ 7f5

d ➔ 2ce
c ➔ c16

Untrusted

Proxy

41b 7f5 2ce

Proxy

9ea c16

Key-Value Store

41b
9ea
7f5
c16
2ce
Shortstack Design Principle #2

Partition Proxy state by Keys

Trusted

Client

Proxy

a → 41b,9ea
b → 7f5

d → 2ce
c → c16

Client

Proxy

Proxy

41b
7f5
2ce

Proxy

9ea
c16

Untrusted

Key-Value Store

41b
9ea
7f5
c16
2ce
Shortstack Design Principle #3

Client

Proxy

**41b**
**7f5**
**2ce**

Proxy

**9ea**
**c16**

Proxy

**2ce**
**c16**

Proxy

**41b**
**9ea**
**7f5**
**c16**
**2ce**

Key-Value Store

Untrusted
Shortstack Design Principle #3

Query Generation over all Keys

Trusted

Untrusted

Key-Value Store

- 41b
- 9ea
- 7f5
- c16
- 2ce

Client

Proxy

- a → 41b 9ea
- b → 7f5

Proxy

- 41b 7f5 2ce

Proxy

- 9ea c16

Proxy

- d → 2ce
- c → c16
Shortstack Design Principle #3

Query Generation over all Keys

Example to provide intuition

trusted

Client

Proxy

41b 7f5 2ce

41b 9ea

7f5

2ce

Untrusted

Key-Value Store

41b

9ea

7f5

c16

2ce

Client

a ➞ 41b 9ea

b ➞ 7f5

d ➞ 2ce

c ➞ c16
Shortstack Design Principle #3

Query Generation over all Keys

Example to provide intuition

[Diagram showing a network of proxies, a key-value store, and client interactions with keys 41b, 7f5, 2ce, 9ea, c16, and 2ce.]
Shortstack Design Principle #3

Query Generation over all Keys

Example to provide intuition

Input Distribution

Client

Proxy

a ➔ 41b 9ea
b ➔ 7f5

d ➔ 2ce
c ➔ c16

Proxy

41b | 7f5 | 2ce

Proxy

9ea | c16

Untrusted

Key-Value Store

41b
9ea
7f5
c16
2ce
Shortstack Design Principle #3

Query Generation over all Keys

Example to provide intuition
Shortstack Design Principle #3

Query Generation over all Keys

Example to provide intuition
Shortstack Design Principle #3

Query Generation over all Keys

Example to provide intuition
Shortstack Design Principle #3

Query Generation over all Keys

Example to provide intuition

1. Client interacts with Proxy.
   - a → 41b 9ea
   - b → 7f5

2. Proxy interacts with Proxy.
   - d → 2ce
   - c → c16

3. Proxy interacts with Key-Value Store.
   - 41b
   - 9ea
   - 7f5
   - c16
   - 2ce
Shortstack Design Principle #3

Query Generation over all Keys

trusted

Untrusted

Proxy

Proxy

Proxy

Proxy

Client

Client

Proxy

Proxy

Proxy

Key-Value Store

41b

9ea

7f5

c16

2ce
Shortstack Design Principle #3

Query Generation over all Keys

Trusted

Untrusted

Key-Value Store

Output Distribution

Client

Proxy

a 41b 9ea
b 7f5

Proxy

d 2ce
c c16

Proxy

41b 7f5 2ce

Client

a b c d

a b c d

a b c d
Shortstack Design Summary: Logical

<table>
<thead>
<tr>
<th>Proxy</th>
<th>Proxy</th>
<th>Proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>a ➔ 41b 9ea</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b ➔ 7f5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41b</td>
<td>7f5</td>
<td>2ce</td>
</tr>
<tr>
<td>d ➔ 2ce</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c ➔ c16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9ea</td>
<td>c16</td>
<td></td>
</tr>
</tbody>
</table>

Key-Value Store:
- 41b
- 9ea
- 7f5
- c16
- 2ce
Shortstack Design Summary: Logical
Shortstack Design Summary: Logical

Layer 1
- Trusted
- Query Generation
- Client

Layer 2
- Proxy State
- Proxy
- Untrusted
- Key-Value Store
- 41b
- 9ea
- 7f5
- c16
- 2ce

Client

Client
Shortstack Design Summary: Logical

Layer 1
- Query Generation

Layer 2
- Proxy State

Layer 3
- Query Execution

Trusted

Client

Untrusted

Key-Value Store
- 41b
- 9ea
- 7f5
- c16
- 2ce
Shortstack Design Summary: Logical

Client

Query Generation

Proxy State

Query Execution

Layer 1

Layer 2

Layer 3

Trusted

Untrusted

Key-Value Store
Shortstack Design Summary: Logical

Client

Layer 1

Layer 2

Layer 3

Query Generation

Proxy State

Query Execution

Untrusted

Key-Value Store

Trusted

Randomly Routed
Shortstack Design Summary: Logical

Client

Randomly Routed

Layer 1

Routed by Key

Layer 2

Layer 3

Query Generation

Proxy State

Query Execution

Untrusted Key-Value Store

Trusted
Shortstack Design Summary: Logical

- **Key-Value Store**
  - **Client** (Trusted)
  - **Client** (Untrusted)
  - **Query Generation** (Layer 1)
    - Randomly Routed
  - **Proxy State** (Layer 2)
    - Routed by Key
  - **Query Execution** (Layer 3)
    - Routed by Replica
  - **Untrusted Key-Value Store**
Shortstack Fault Tolerance

Trusted

Layer 1

Layer 2

Layer 3

Untrusted

Key-Value Store
Shortstack Fault Tolerance

Key-Value Store
Shortstack Fault Tolerance

L3 failures: Redistribute requests across remaining servers
Shortstack Fault Tolerance

L3 failures: Redistribute requests across remaining servers
L1 and L2 failures handled through chain replication
Shortstack Fault Tolerance

L3 failures: Redistribute requests across remaining servers

L1 and L2 failures handled through chain replication
Shortstack Fault Tolerance

L3 failures: Redistribute requests across remaining servers

L1 and L2 failures handled through chain replication
Shortstack Fault Tolerance

L3 failures: Redistribute requests across remaining servers
L1 and L2 failures handled through chain replication
Logical ➔ Physical Design

Trusted

Client ➔ L1A Head ➔ L1A Mid ➔ L1A Tail ➔ L1B Head ➔ L1B Mid ➔ L1B Tail ➔ L1C Head ➔ L1C Mid ➔ L1C Tail ➔ L2A Head ➔ L2A Mid ➔ L2A Tail ➔ L2B Head ➔ L2B Mid ➔ L2B Tail ➔ L2C Head ➔ L2C Mid ➔ L2C Tail ➔ L3A ➔ L3B ➔ L3C ➔ Physical Server 3

Untrusted

Key-Value Store ➔ Physical Server 1 ➔ Physical Server 2 ➔ Physical Server 3
Logical → Physical Design

Client → L1A Head → L1A Mid → L1A Tail → L1B Head → L1B Mid → L1B Tail → L1C Head → L1C Mid → L1C Tail → L2A Head → L2A Mid → L2A Tail → L2B Head → L2B Mid → L2B Tail → L2C Head → L2C Mid → L2C Tail → L3A → L3B → L3C → Key-Value Store

Trusted

Client

Physical Server 1

Physical Server 2

Physical Server 3

Untrusted
Logical ➔ Physical Design

Client ➔ Trusted ➔ Physical Design ➔ Untrusted ➔ Key-Value Store

Physical Server 1 ➔ L1A Head ➔ L1A Mid ➔ L1A Tail ➔ L1B Head ➔ L1B Mid ➔ L1B Tail ➔ L1C Head ➔ L1C Mid ➔ L1C Tail

Physical Server 2 ➔ L2A Head ➔ L2A Mid ➔ L2A Tail ➔ L2B Head ➔ L2B Mid ➔ L2B Tail ➔ L2C Head ➔ L2C Mid ➔ L2C Tail

Physical Server 3 ➔ L3A ➔ L3B ➔ L3C
Logical ➔ Physical Design

Shortstack needs 3 physical servers for handling 2 failures
Shortstack needs 3 physical servers for handling 2 failures.

Shortstack needs $f+1$ physical servers for handling $f$ failures.
This work

Challenges with Centralized Oblivious Data Access Systems
1. Insecure or unavailable during failures
2. Scalability bottleneck

Design of Shortstack:
Distributed, Fault-tolerant, Oblivious Data Access System

Security model:
Enables formal study of Distributed, Fault-tolerant, Oblivious Data Access systems
Security Model
Existing security models do not capture failures
Security Model

Existing security models do not capture failures

New, General, Security Model
Security Model

Existing security models do not capture failures

New, General, Security Model

Powerful adversary that cause arbitrary failures (bounded number) at arbitrary times
Powerful adversary that cause arbitrary failures (bounded number) at arbitrary times

+ Output distribution independent of input distribution (Oblivious data access guarantee)

Existing security models do not capture failures

New, General, Security Model
Security Model

Existing security models do not capture failures

New, General, Security Model

- Powerful adversary that cause arbitrary failures (bounded number) at arbitrary times
- Output distribution independent of input distribution (Oblivious data access guarantee)

(Formal definitions and proof of Shortstack security in paper)
Evaluation
Evaluation

• ShortStack end-to-end implementation open-sourced
  • Requires no modifications to the server side
Evaluation

- **ShortStack end-to-end implementation open-sourced**
  - Requires no modifications to the server side
- **Evaluation on EC2 with Redis as the key-value store**
  - YCSB-A and YCSB-C workloads (more details in the paper)
Evaluation

- ShortStack end-to-end implementation open-sourced
  - Requires no modifications to the server side
- Evaluation on EC2 with Redis as the key-value store
  - YCSB-A and YCSB-C workloads (more details in the paper)
- Goal: Demonstrating throughput scalability with number of physical servers
Evaluation

- **ShortStack end-to-end implementation open-sourced**
  - Requires no modifications to the server side

- **Evaluation on EC2 with Redis as the key-value store**
  - YCSB-A and YCSB-C workloads (more details in the paper)

- **Goal:** Demonstrating **throughput scalability with number of physical servers**

![Throughput Scalability Graph](chart.png)
Evaluation

- ShortStack end-to-end implementation open-sourced
  - Requires no modifications to the server side
- Evaluation on EC2 with Redis as the key-value store
  - YCSB-A and YCSB-C workloads (more details in the paper)
- Goal: Demonstrating throughput scalability with number of physical servers
Evaluation

- ShortStack end-to-end implementation open-sourced
  - Requires no modifications to the server side
- Evaluation on EC2 with Redis as the key-value store
  - YCSB-A and YCSB-C workloads (more details in the paper)
- Goal: Demonstrating **throughput scalability with number of physical servers**

![Graph showing throughput scalability with number of physical servers]

Throughput with Centralized proxy
Evaluation

• ShortStack end-to-end implementation open-sourced
  • Requires no modifications to the server side
• Evaluation on EC2 with Redis as the key-value store
  • YCSB-A and YCSB-C workloads (more details in the paper)
• Goal: Demonstrating throughput scalability with number of physical servers
Evaluation

• ShortStack end-to-end implementation open-sourced
  • Requires no modifications to the server side
• Evaluation on EC2 with Redis as the key-value store
  • YCSB-A and YCSB-C workloads (more details in the paper)
• Goal: Demonstrating **throughput scalability with number of physical servers**

![Throughput graph](image)
Evaluation

- **ShortStack end-to-end implementation open-sourced**
  - Requires no modifications to the server side
- **Evaluation on EC2 with Redis as the key-value store**
  - YCSB-A and YCSB-C workloads (more details in the paper)
- **Goal:** Demonstrating **throughput scalability with number of physical servers**
Evaluation

• **ShortStack end-to-end implementation open-sourced**
  • Requires no modifications to the server side

• **Evaluation on EC2 with Redis as the key-value store**
  • YCSB-A and YCSB-C workloads (more details in the paper)

• **Goal: Demonstrating throughput scalability with number of physical servers**
• Many additional results in the Paper…
  1. System Scalability
  2. Fault Tolerance
  3. Latency
  4. Bottlenecks in each layer
  5. Skewed workloads
Conclusion

Challenges with Centralized Oblivious Data Access Systems
1. Insecure or unavailable during failures
2. Scalability bottleneck

Design of Shortstack:
Distributed, Fault-tolerant, Oblivious Data Access System

Security model:
Enables formal study of Distributed, Fault-tolerant, Oblivious Data Access systems

https://github.com/pancake-security/shortstack