SPORC

Group Collaboration using Untrusted Cloud Resources

Ariel J. Feldman, William P. Zeller, Michael J. Freedman, Edward W. Felten
Cloud deployment: pro & con

For user-facing applications:
(e.g. word processing, calendaring, e-mail, IM)

Cloud deployment is attractive
• Scalable, highly available, globally accessible
• Real-time collaboration

But, there’s a price…

Must trust the cloud provider for confidentiality and integrity
SPORC goals

Practical cloud apps
• Flexible framework
• Real-time collaboration
• Work offline

*Untrusted* servers
• Can’t read user data
• Can’t tamper with user data without risking detection
• Clients can recover from tampering
Making servers untrusted

SPORC Server’s limited role:

- Storage
- Ordering msgs

Client 1
- App logic
- Copy of state

Client 2
- App logic
- Copy of state

Client
Problem #1: How do you keep clients’ local copies consistent?
(esp. with offline access)
Problem #2: How do you deal with a malicious server?

Client 1
- App logic
- Copy of state

Client 2
- App logic
- Copy of state

Server
- Encrypted state

Client
Keeping clients in sync

Operational transformation (OT) [EG89]
(Used in Google Docs, EtherPad, etc.)

OT can sync arbitrarily divergent clients
Dealing with a malicious server

Digital signatures aren’t enough

Server can **equivocate**

fork* consistency \[\text{[LM07]}\]

- Honest server: linearizability
- Malicious server: Alice and Bob detect equivocation after exchanging 2 messages
- Embed history hash in every message

**Server can still fork the clients, but can’t unfork**
System design

Client app

Local state

SPORC lib
System design

Client app

Local state

Committed Pending

fork* consistent causally consistent

SPORC lib
System design

Client app

Local state

Committed
Pending

Encrypt & sign

Server

Encrypted state

SPORC lib
System design

Client app
- Local state
- Committed
- Pending

SPORC lib

Compare history hashes

Verify & decrypt

Server
- Encrypted state

Client
System design

Client app

Local state

SPORC lib

Committed Pending

Decrypt & verify

Server

Encrypted state

Client

SPORC: Group Collaboration using Untrusted Cloud Resources — OSDI 10/5/10
System design

Client app

Local state

Committed

Pending

SPORC lib

Server

Encrypted state

Client

T
Access control

Challenges
• Server can’t do it — it’s untrusted!
• Preserving causality
• Concurrency makes it harder

Solutions
• Ops encrypted with symmetric key shared by clients
• ACL changes are ops too
• Concurrent ACL changes handled with barriers
Adding a user

Group members:

- Alice
- Bob
- Charlie

ModifyUserOp

Add “Charlie”

$E_{\text{Charlie\_pk}}(k)$
Removing a user

Group members:

Alice

Bob

Charlie

Server

ModifyUserOp

Rm “Charlie”

$E_{alice\_pk}(k')$

$E_{bob\_pk}(k')$

$E_k(k)$
Barriers: dealing with concurrency

Clients check on the server

Group members:
- Alice
- Bob
- Charlie
- Eve

Server

ModifyUserOp
Rm “Charlie”
$E_{k1}(k)$

ModifyUserOp
Rm “Eve”
$E_{k2}(k)$

ModifyUserOp
Rm “Charlie”
$E_{k1}(k)$

ModifyUserOp
Rm “Eve”
$E_{k2}(k1)$
Recovering from a fork

Can use OT to resolve malicious forks too
Implementation

Client lib + generic server

App devs only need to define ops and provide a transformation function

Java CLI version + browser-based version (GWT)

Demo apps: key value store, browser-based collaborative text editor
Evaluation

Setup

• Tested Java CLI version
• 8-core 2.3 GHz AMD machines
  • 1 for server
  • 4 for clients (often >1 instance per machine)
• Gigabit LAN

Microbenchmarks

• Latency
• Server throughput
• Time-to-join (in paper)
Latency

Low load
(1 client writer)

High load
(all clients are writers)
Latency

Low load
(1 client writer)

High load
(all clients are writers)
Server throughput

![Graph showing server throughput with payload size (KB) on the x-axis and throughput (MB/s) on the y-axis. The graph includes two lines representing MB/s and ops/s.]
Conclusion

Practical cloud apps + untrusted servers

Operational transformation + fork* consistency

Dynamic access control and key distribution

Recovery from malicious forks
Thank you

Questions?

ajfeldma@cs.princeton.edu

Comparison with Depot

<table>
<thead>
<tr>
<th>Feature</th>
<th>SPORC</th>
<th>Depot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistency with malicious servers</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Consistency with malicious clients</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>Fork recovery</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Work offline</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Dynamic access control</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Confidentiality and key distribution</td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

Depot exposes conflicts, but leaves it to the app to resolve them

Future work: SPORC + Depot? ;-)}
Time-to-join

![Graph showing the relationship between Client time-to-join (s) and Number of committed operations for different applications: Text Editor (w/ pending), Key-Value (w/ pending), Text Editor, and Key-Value. The graph illustrates linear trends with increasing time as the number of operations committed increases.](image)