SPORC

Group Collaboration using Untrusted Cloud Resources

Ariel J. Feldman, William P. Zeller, Michael J. Freedman, Edward W. Felten

PRINCETON UNIVERSITY
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

App logic
State

Server

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 [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

Encrypt
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

Committed
Pending

SPORC lib

Decrypt & verify

Server

Encrypted state

Client
System design

Client app

Local state

Committed Pending

SPORC lib

Server

Encrypted state

Client
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

Server

ModifyUserOp
Add “Charlie”
$E_{\text{Charlie\_pk}}(k)$

Group members:
Alice
Bob
Charlie
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

SPORC: Group Collaboration using Untrusted Cloud Resources — OSDI 10/5/10
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)
Server throughput

![Graph showing server throughput vs payload size (KB)](image)

- Throughput (MB/s) vs Payload size (KB)
- Operations per second (ops/s) vs Payload size (KB)

Throughput (MB/s): Red line
Operations per second (ops/s): Blue line

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