Reliable, Consistent, and Efficient Data Sync for Mobile Apps

Younghwan Go*, Nitin Agrawal,
Akshat Aranya, and Cristian Ungureanu

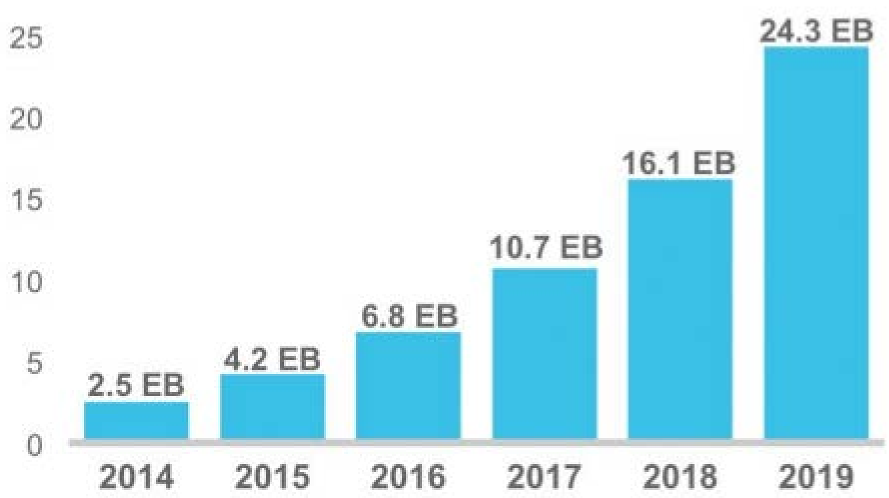
NEC Labs. America KAIST*

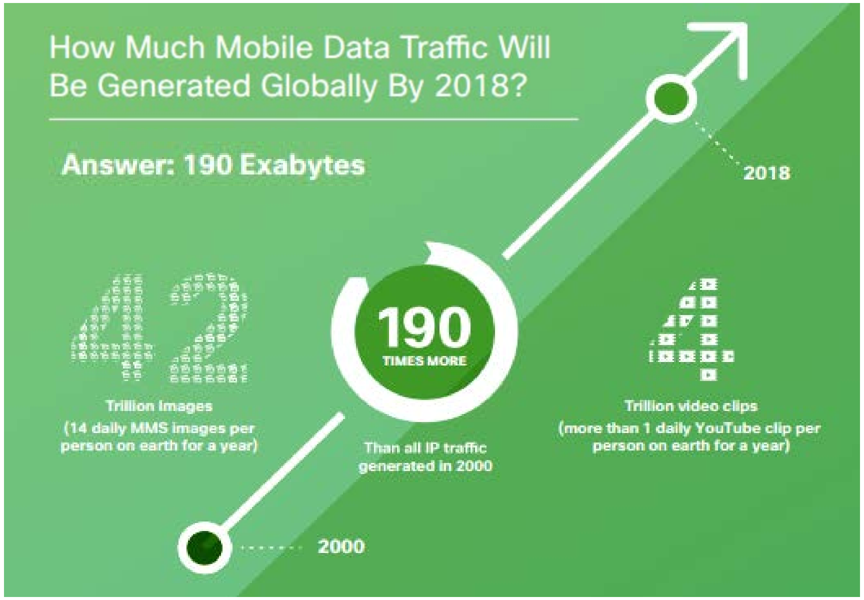
NEC Laboratories America
KAIST

Increase in Data-centric Mobile Apps

- Massive growth in mobile data traffic [Cisco VNI Mobile 2014]
  - 24.3 Exabytes per month by 2019
  - 190 Exabytes of mobile traffic generated globally by 2018

    = 42 trillion images, 4 trillion video clips



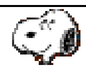


Difficulty in Building Data-centric Apps

- Reliability: transparent failure handling

Look!
My data is
corrupted!

- Consistency: concurrent updates, sync atomicity

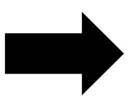
RowId	Name	Photo
1	Snoopy	
2	Lucy	

→ Structured data

→ Unstructured data

Row ID	Col	Obj
ID_{row}	name	file

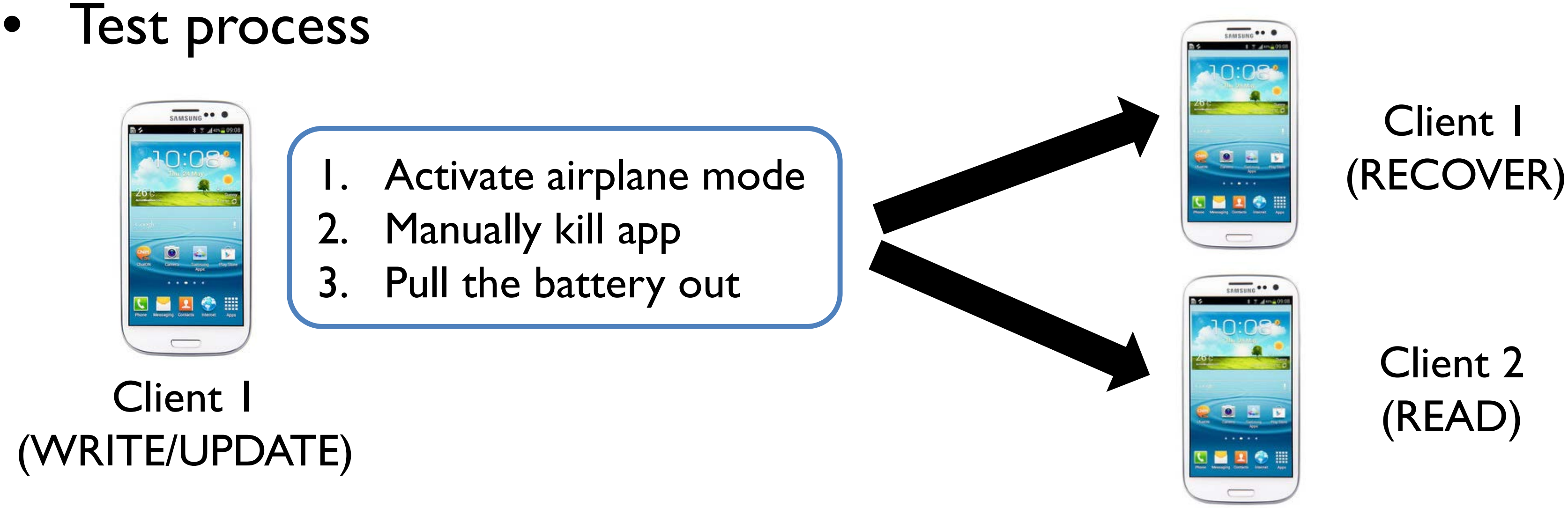
- Efficiency: minimize traffic/battery usage



010111001
110110111

Mobile App Study on Reliability

- Study mobile app recovery under failures
  - Network disruption, local app crash, device power loss
  - Analyze recovery when failed during write/update
- Test 15 apps that use tables and objects
  - Independent or existing sync services (e.g., Dropbox, Parse, Kinvey)
- Test process



Current Mobile Apps are not Reliable!

- Disruption recovery
  - Loss of data if app/notification closed during disruption
  - No notification of sync failure
  - Manual re-sync creates multiple copies of same note
- Crash recovery
  - Partial object created locally without sync
  - Corrupted object synced and spread to second client
- Additional observations
  - No app correctly recovered from crash at object update
  - Many apps simply disable object update capability altogether

More details of the study can be found in our paper ☺

Goals of Sync as a Service

- Reliability
  - User can always sync to the latest data
  - User's update is guaranteed to be synced to server
- Consistency
  - Data can always return to a consistent state even after failures
  - Inter-dependent structured/unstructured data are synced atomically
- Efficiency
  - Minimum mobile data traffic is generated for sync/recovery
  - Device's overall network radio usage is reduced to save battery

Outline

- Introduction
- Mobile app study on reliability
- Simba Client Design
- Evaluation
- Conclusion

Simba: Data-sync Service for Mobile Apps

- High-level programming abstraction
  - CRUD-like interface for easy development
  - Unify tabular and object data

- Transparent handling of data syncs and failures
  - Failure detection & recovery at network disruption and crash
  - Guarantee atomic sync of tabular and object data

- Resource frugality with delay-tolerance and coalescing
  - Delay sync messages to be clustered
  - Reduce number of network messages & radio usage

Writing a Photo App with Simba

- Create a photo album

```
createTable("album", "name VARCHAR, photo OBJECT", FULL_SYNC);
```

- Register read/write sync

```
registerReadSync("album",600,0,3G);   // period=10min, pref=3G
registerWriteSync("album",300,0,WIFI);// period=5min, pref=WiFi
```

- Add a new photo

```
objs = writeData("album", {"name=Snoopy"}, {"photo"});
objs[0].write(photoBuffer); // write object data
```

- Retrieve stored photo

```
cursor = readData("album", {"photo"}, "name=?", {"Snoopy"});
mis = cursor.getInputStream().get(0); // inputstream for object
mis.read(buffer); // read object data into buffer
```

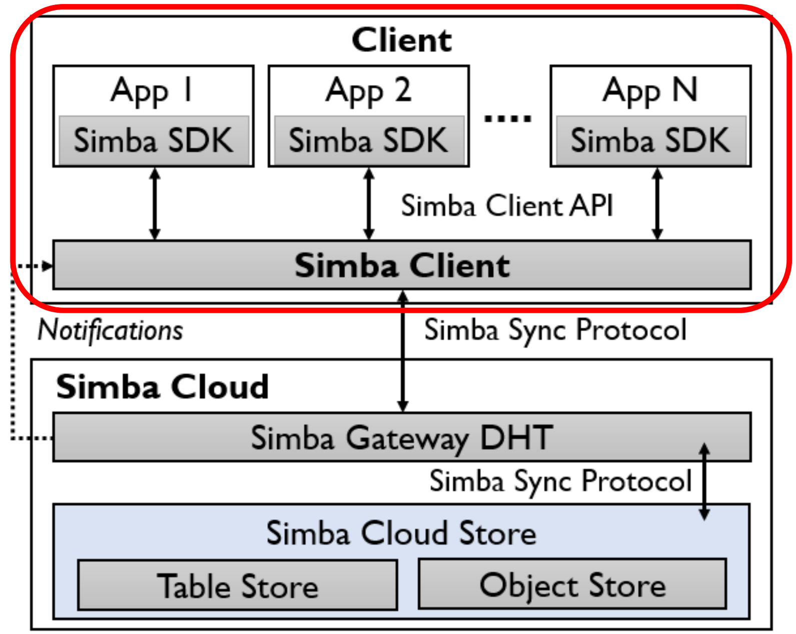
Writing a Photo App with Simba

- Conflict resolution

```
beginCR("album");
rows = getConflictedRows("album");
for (row; rows; next row) {
    // choice = MINE, THEIRS, OTHERS
    resolveConflict("album", row, MINE);
}
endCR("album");
```

Overall Architecture

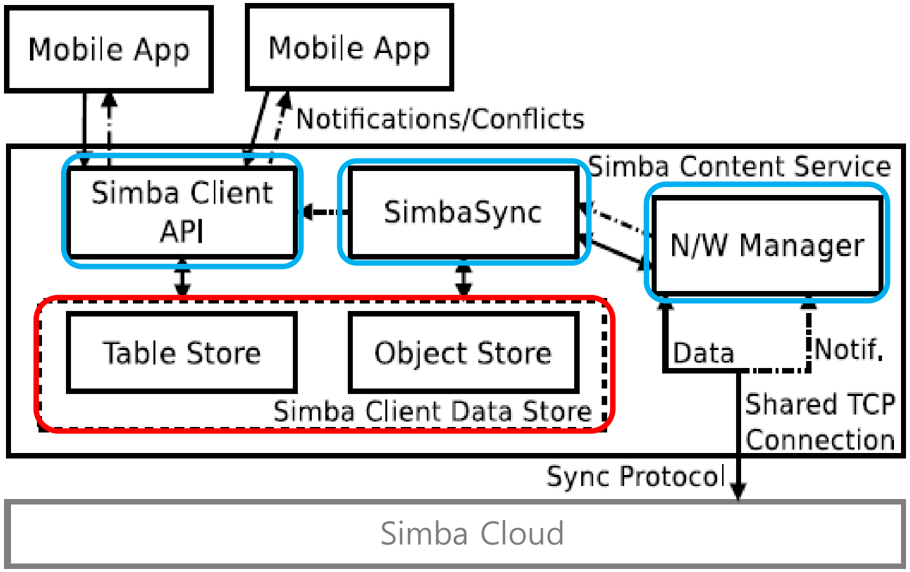
- Reliable data sync between sClient ↔ sCloud
  - Simba Cloud (sCloud)
    - Manage data across multiple apps, tables, and clients
    - Respond to sClient's sync request
    - Push notifications to sClient
  - Version-based Sync Protocol
    - Row-level consistency
    - Unique id per row, ID_{row}
    - One version per row, V_{row}



Simba Cloud paper to be presented at EuroSys 2015!
"Simba: Tunable End-to-End Data Consistency for Mobile Apps"

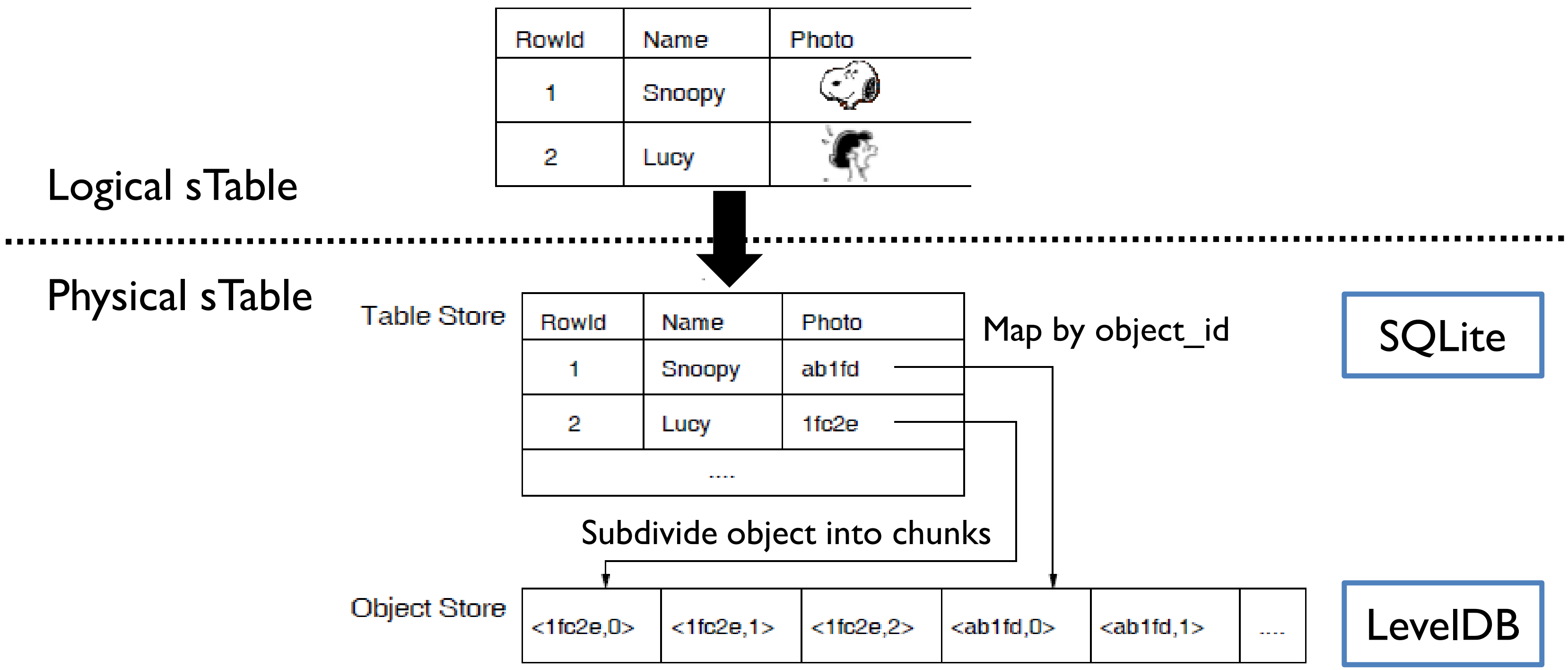
sClient: Simba Content Service

- Simba Client API (sClientLib)
  - Interface to access table and object data for apps
  - Upcall alerts for events (new data, conflict) to apps
- SimbaSync
  - Manage fault-tolerance, data consistency, row-level atomicity
- N/W Manager
  - Send/receive sync messages, receive notifications
- Simba Client Data Store



Simba Client Data Store

- We don't want half-formed data to appear on our phone!
- Simba Table (sTable)
  - Unified table store for tabular and object data



Simba Local States

- Include additional local states to determine:
  - Health of data (latest vs. updated)
  - Sync readiness (object closed after update)
  - Failure state (sync in progress after network disruption)
  - Recovery actions (retry, reset, recover corrupted objects, etc.)
- Simba local states

		Update in tab | obj data		End of obj update	Sync in progress	Row in conflict		
Row ID	Version	$Flag_{TD}$	$Flag_{OD}$	$Count_{OO}$	$Flag_{SP}$	$Flag_{CF}$	Name	Photo
ID_{row}	V_{row}	0/1	0/1	0/1/../n	0/1	0/1	"Snoopy"	object_id

  - Dirty Chunk Table (DCT): updated chunk ids per object

Handling Network Failures

- Move to a consistent state after network disruption
- Detect & recover in the middle of sync
  - Consult state upon network disruption
  - Recovery policy dependent on server response (RC_T, RC_O, RU_T, RU_O)
    - No op, normal operation, retry, reset & retry, roll forward
- Upstream sync example

State at network disruption	Implication	Recovery Policy	Action
[SP=1] before sync response	Missed response	Reset & retry	SP=0, TD=1, OD=1 if ∃DCT

- Downstream sync example

State at network disruption	Implication
[RU_O=1] after sync response	Partial response

→

TD	OD	RU_T	Recovery Action
*	*	*	Delete entry, resend downstream sync request

Handling App/Device Failures

- Roll back/forward to a consistent state after crash
- Recovery policy dependent on local states
  - $Flag_{TD}, Flag_{OD}, Count_{OO}, Flag_{SP}, Flag_{CF}, DCT$
- Recover from a crash during sync

TD	OD	OO	SP	CF	Recovery Action
0	0	=0	1	-	Restart upstream sync (SP = 0, TD = 1, OD = 1 if ∃DCT)

- Recover from a crash at update

TD	OD	OO	SP	CF	Recovery Action
1	0	>0	0	0	Start upstream sync (OO = 0)
*	1	>0	0	0	Torn row! Retrieve consistent row version from sCloud (TD = 0, OD = 0, OO = 0)

Evaluation

- Evaluation goals
  - Does Simba provide transparency to apps?
  - Does Simba perform well for sync and local I/O?
- Evaluation setup
  - sClient
    - Galaxy Nexus (Android 4.2)
    - Nexus 7 (Android 4.2)
  - sCloud
    - 2 Intel Xeon servers: 16-core (2.2GHz), 64GB DRAM, 8 7200RPM 2TB disk
    - 4 VMs on each sCloud: 4 core, 8GB DRAM, one disk
  - WiFi: 802.11n (WPA)
  - Cellular: 4G LTE (KT, LGU+, AT&T)

App Development with Simba

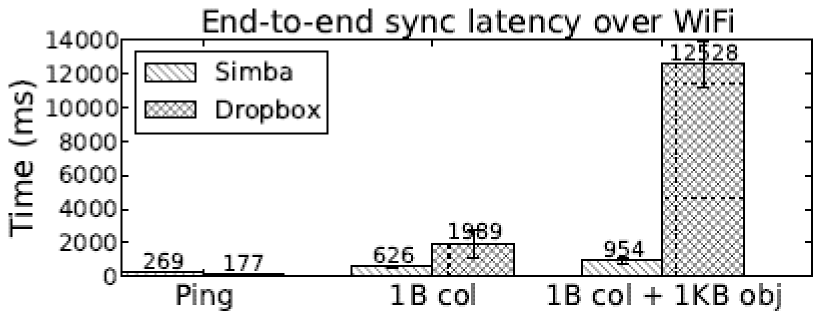
- Simple and easy app development with Simba

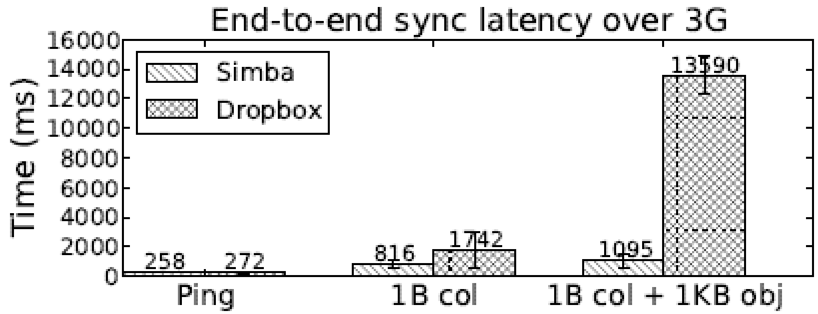
App	Description	Total LoC	Simba LoC
Simba-Notes	"Rich" note-taking with embedded images and media	4,178	367
HbeatMonitor	Monitor and record a person's heart rate, cadence and altitude using Zephyr heartbeat sensor	2,472	384
CarSensor	Record car engine's RPM, speed, engine load, and etc using Soliport OBD2 sensor	3,063	384
Simba-Photo	Photo-sync app with write/update/read/delete operations on tabular and object data	527	170

- Building a photo app with existing sync service (Dropbox)
  - No inter-operation of table and object
  - No support for row-level atomicity (only column-level!)
  - No detection & recovery of torn rows

Sync Performance

- End-to-end sync latency for "1B col" & "1B col + 1KB obj"
- Test method
  - Client 1 updates for sync → client 2 receives update
  - Clients (Korea), sCloud (Princeton), Dropbox server (California)



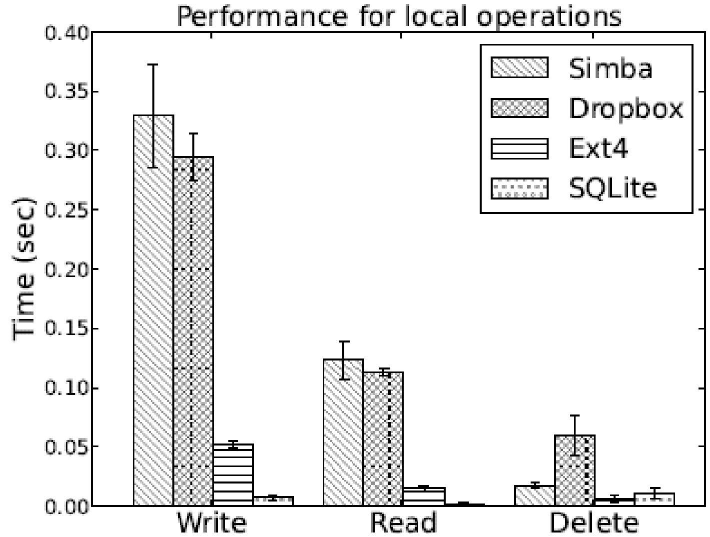


- Results
  - Network latency: small component of total sync latency
  - Simba performs well compared to Dropbox in all cases

NEC Laboratories America KAIST

Local I/O Performance

- Time to write/read/delete one row with 1MB object



- ~10% slower than Dropbox for write/read
  – IPC overhead between Simba-app and sClient
- Better than Dropbox for delete
  – Lazy deletion: marked for delete → delete after sync completion

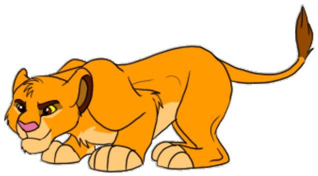
Conclusions

- Building data-centric mobile app should be transparent
  - Mobile app developers should focus on implementing app core logic
  - Require service that handles complex network and data management
- Simba: reliable, consistent, and efficient data-sync service
  - Unified sTable and API for managing tabular and object data
  - Transparent handling of data syncs and failures
  - Resource frugality with delay-tolerant coalescing of sync messages
- Practical for real-world usage
  - Easy app development/porting with CRUD-like API
  - Sync performance comparable to existing services
  - Minimum local I/O overhead

Thank you!

Simba source: https://github.com/SimbaService/Simba

Project homepage: http://www.nec-labs.com/~nitin/Simba



NEC Laboratories America KAIST

Related Works

- Data sync services
  - Parse, Kinvey, Bayou, Mobius [MobiSys'12]: support table sync
  - LBFS [SOSP'01]: support file sync
  - Do not provide sync service for both tables and objects
- Failure tolerance
  - ViewBox [FAST'14]: guarantee consistency of local data at crash
  - Works for files in desktop FS
- Storage unification
  - TableFS [ATC'13]: separate storage pools for metadata and files
  - KVFS [FAST'13]: store file and metadata in a single key-value store
  - Consider integration without network sync or a unified API

Balancing Sync Efficiency & Transparency

- In-memory vs. persistent DCT
  - Sync only updated chunks for each object during sync
  - In-memory DCT lost after crash: send entire object → inefficient!
  - Persist DCT to prevent re-syncing entire, potentially large objects
- In-place vs. out-of-place update
  - Recover a torn (corrupted) row with data from the consistent state
  - Out-of-place: local state + I/O overhead for common-case operation
  - In-place: retrieve consistent version of row from sCloud

Operation	Method	Throughput (MB/s)
Update	In-place	2.29 ± 0.08
	Out-of-place	1.37 ± 0.04
Read	In-place	3.94 ± 0.04
	Out-of-place	3.97 ± 0.07

↑ 69%