Accelerating Mobile Applications through Flip-Flop Replication

Gordon et al. 2015

Presented by John T Hill IV, 2022
University of Pittsburgh -- ECE 2195 Special Topics: Edge Computing

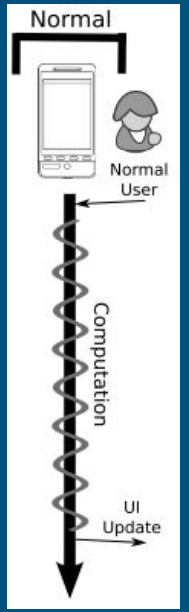
Overview

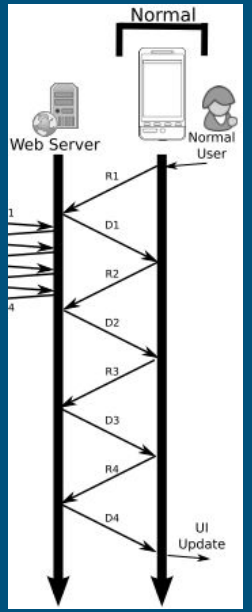
- Challenges with offloading mobile computation & dynamic cost evaluation
- *Tango* introduces scaled-redundancy to flip-flop between mobile & remote processing
- Comparing Tango with Android Application Benchmarks
- Follow-up advances
- Conclusions

Mobile Computing Workload (1)

- Mobile devices limitations
  - Has poorer computational performance due to low-power circuitry
  - Has poorer network performance due to over-the-air comms challenges

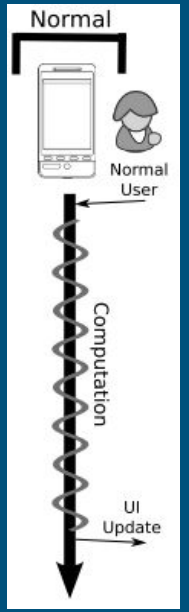
- Offloading computation could help with compute-intensive performance, but challenging to find good partitioning moments

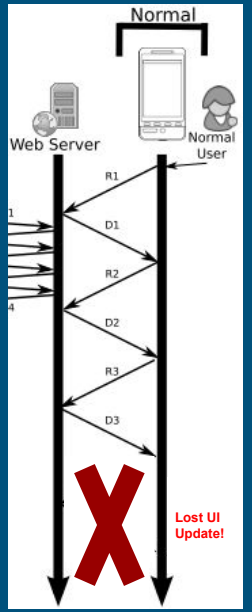




Mobile Computing Workload (2)

- Over-the-air comms with lost connectivity causes slower (or lost) UI responsiveness
  - In reality no datagram is sent as a single payload, so multiple request/response pairs are always needed
- The challenge is to find a dynamic way to support both of these problems



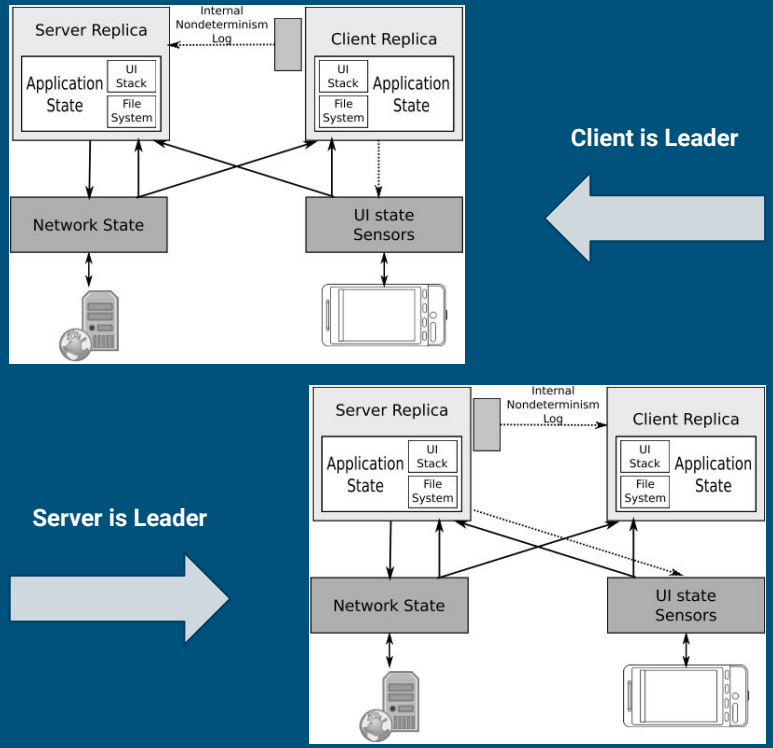


"The fundamental problem is that the right place to run a program depends on many factors."

How Tango Reduces Application Latency

1. Introduce a replica on both mobile & remote platforms, allowing either to send output to the user
2. Use *deterministic replay* to ensure the replicas perform the same computations & outputs
3. Share role of logging non-deterministic inputs from external sources & externalizing outputs
4. Replicate I/O sources to reduce amount of non-determinism
5. Allow replicas to log internal sources of non-determinism (thread & event scheduling, time queries, etc.)

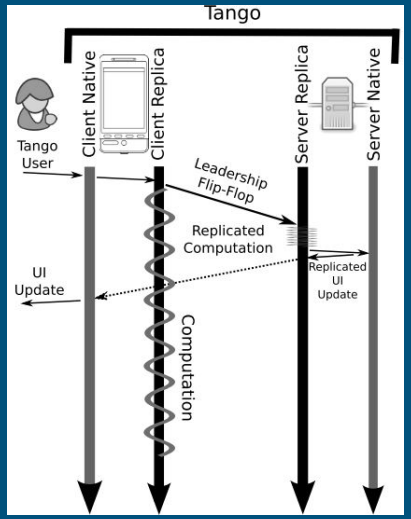
Architectural Overview



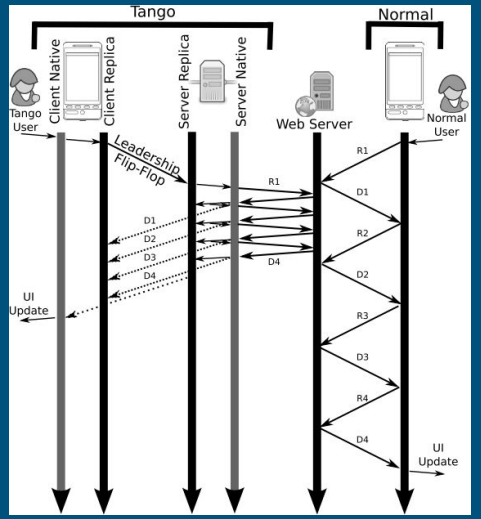
- Replicated targets run in Dalvik VMs executing the same application
  - Enforces deterministic co-execution independent of native ISAs; whatever source finishes first becomes “leader”
  - Supported by replicating native methods that the VM invokes (e.g., `java.math` functions)
  - Each application has frequent UI stack interactions; provides a “finish-line”
- External non-replicated parts (e.g., I/O) are executed on the replica first until native method is called; resulting state and return transferred to follower replica
- Internal non-replicated parts (e.g., scheduling decisions) are controlled by Tango itself to enforce determinism
  - Events elevate replica to leader status, thus state is transferred to follower replica

Example

- **User interacts with application UI; events are broadcast to both replicas**
  - Client replica is Leader because it receives UI inputs first
  - Server replica gets inputs as a pipeline before asking
- **Application enters compute-intensive phase**
  - Server replica quickly catches up to Client compute stage, flip-flops leadership
  - Server eventually calls UI methods, send results directly to client
- **Application enters network-intensive phase**
  - Server replica assumes leadership since it has faster comms (running at data center)
  - Responses sent in pipeline fashion like compute-intensive phase (one-way network delay!)
  - With small amount of computational burden, Client replica catches up, flip-flops leadership while waiting on remote responses



Compute Intensive Phase



Network Intensive Phase
(with & without Tango)

Implementation (1) - Threads & Native Methods

- Utilize *single managed threads* with a deterministic scheduling algorithm
  - Thread is managed any time it directly interacts with Dalvik VM state
  - Context switching of threads is round-robin

- When Java thread invokes a native method (via JNI), Tango moves thread to unmanaged execution; upon completion, thread is moved back to managed-mode
  - To prevent divergence, both replicas must agree on when an asynchronous event happens
  - Note, native method executions happen on *native threads*; Tango synchronizes these threads so they don't conflict

Implementation (2) - External Nondeterminism

- Proxy threads are run on both replicas to receive external inputs; Leader decides on *when* data arrives within replicated execution stream
- Network transmits are server-replica responsibilities; receives pushed to both replicas; leader decides on timing & results of `poll` & `select`
- UI handled similarly as network, but is client-replica responsibility
- Tango replicates application-specific portion of file system on both replicas by having each update their own local copies
  - External memory (e.g., SD card) are run by native thread on client, so server-replica gets copy
- Leader controls timestamp returns

Implementation (3) - Leadership & Server Faults

- Leader periodically messages Follower asking if it will accept leadership
  - Message contains current point of execution and pauses for response
  - Rejected leadership switch incurs a round-trip time (RTT) delay
- Since switching is cheap, requests are made when current leader has a dependency which incurs a RTT cost, but challenging to do when computation is significant
  - Prediction is supported by tracking how much computation has been completed since previous native method call
- When remote-leader is unable to communicate with client-follower, follower relies on a backup
  - A backup server located near the remote replica captures a log of non-deterministic operations
  - Will continue as non-replicated application when it’s caught up

Evaluation - Methodology

- Macrobenchmarks tested with & without tango on Samsung Galaxy S3 with Android 4.2.2
  - Network latency emulated via USB & netem rules for repeatability
  - No JIT support
- Sudoku & Poker selected as compute-intensive apps
- Hoot, TapTu, Email, Instagram, & Pinterest selected as network-intensive apps
- In addition to timely performance, energy consumption is also considered

Benchmark	Package	Network RTTs
Sudoku	de.georgwiese.sudokusolver	N/A
Poker	com.leslie.cjpokeroddscaculator	N/A
Hoot	com.hootsuite.droid.full	5
TapTu	com.taptu.streams	4
Email	com.android.email	4
Instagram	com.instagram.android	3
Pinterest	com.pintrerest	2-8

Evaluation - Compute Intensive Results

- Measured time from user input to screen update as perceived latency; 10 samples per configuration each
- Tango reduces perceived user latency in all cases compared with baseline
- Performance seems to bottleneck due to client-replica CPU contention

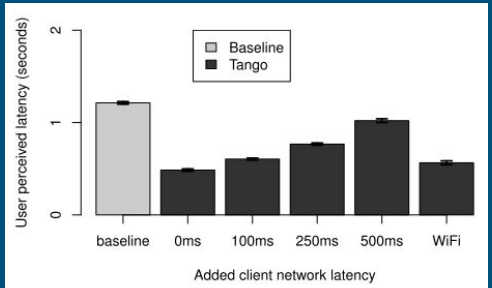


Figure 5: Sudoku performance

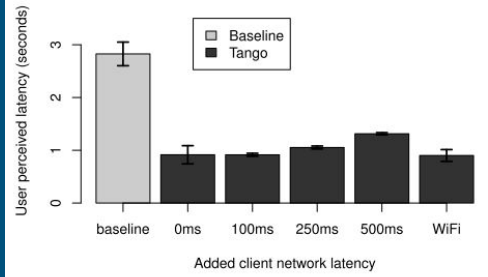
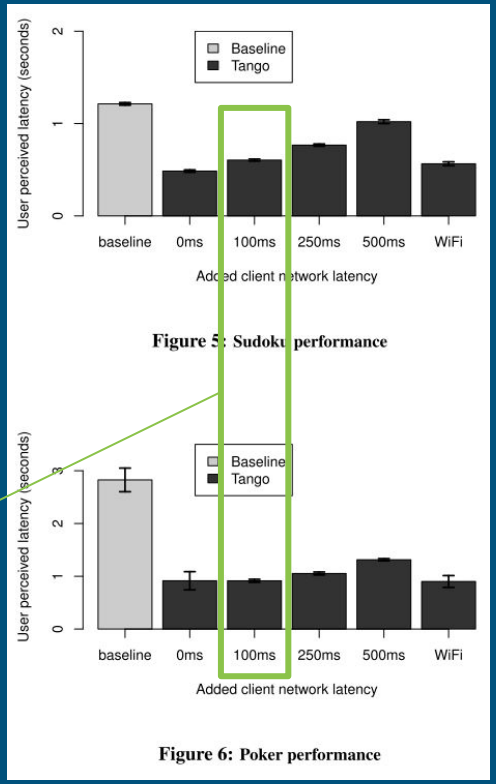


Figure 6: Poker performance

Evaluation - Compute Intensive Results

- Measured time from user input to screen update as perceived latency; 10 samples per configuration each
- Tango reduces perceived user latency in all cases compared with baseline
- Performance seems to bottleneck due to client-replica CPU contention

Typical LTE network delay!



Evaluation - Network Intensive Results

- Measured time from user input to screen update as perceived latency; 14 samples per configuration each
- Tango achieves latency speedup from 1.3-2.6x compared to baselines experiencing equivalent network delays
- Speedup can be attributed to needing fewer network request/response pairings (~1.03/sec compared to ~4.17/sec)

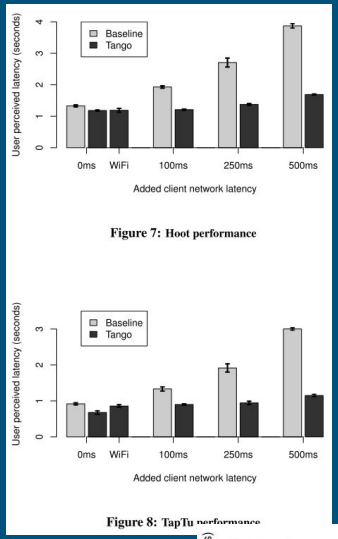


Figure 7: Hoot performance

Figure 8: TapTu performance

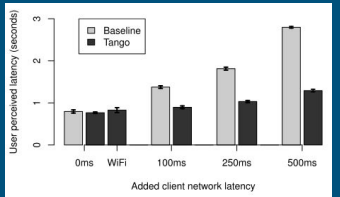


Figure 9: Email performance

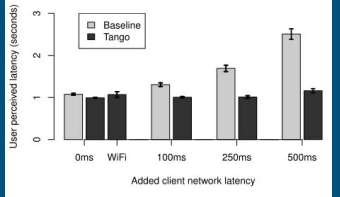


Figure 10: Instagram performance

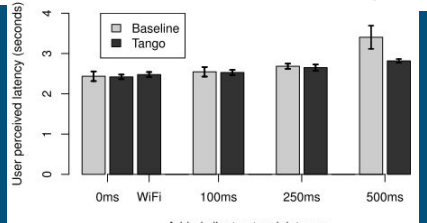


Figure 11: Pinterest performance

Evaluation - Energy Consumption

- Measurement assumes user pauses after screen update (3s for Poker, 1s for rest) plus time to finish all computation on both client & server; 10 samples
- Tango does offer an energy reduction in most cases, but not consistent
- Can be attributed to increased overhead adding to energy cost even though there is reduced completion times

Application	Baseline (J)	Tango (J)	%Change
Sudoku	2.47±.18	2.43±.17	-1.62%
Poker	7.14±.41	5.43±.22	-23.9%
Hoot	2.77±.15	2.46±.21	-11.2%
Taptu	2.14±.23	1.74±.21	-18.7%
Email	1.74±.12	1.78±.18	+2.30%
Instagram	2.26±.16	2.35±.20	+3.98%
Pinterest	4.16±.49	4.06±.30	-2.40%

Lessons Learned (1)

- Retrofitting platforms with Tango is not trivial
  - Using VMs did not always provide clean application partitioning
  - Supporting unmodified applications proved challenging, prevents many applications from gaining benefits
  - Effectively as complex as adding a deterministic hypervisor on top of a GPOS
- Most Android apps execute large numbers of native methods; trend seems to be increasing
  - Challenge is to identify & categorize (and possibly supply deterministic implementations) of appropriate methods
  - Heavy use of WebKit in many applications
- Supporting JIT and Ahead-of-Time compilation (ART) needs engineering effort and porting

Lessons Learned (2)

- Originally Tango was envisioned to support computation-heavy applications; surprisingly supports network-intensive applications!
  - Reduces developer workload because now they don't need to consider RTT costs as strongly in design

- Tango has limitations
  - Dealing with false dependencies among threads (e.g., one client thread sends, another receives) can hinder leadership decisions
  - Serialization of all managed threads impacts computation-heavy applications, prevents full client-side resource utilization
  - Leader-switch decision is still difficult to optimize

Advancements since Tango

- Many “direct” descendant applications!
- DeepWear (Xu2019)
  - Wearable devices have ***even less*** resources than cellular devices, but has DL needs; consider context-aware scheduling, partial offloading, and data streaming
- Web App Computational Offloading for ML (Jeong2018)
  - DNNs have huge computational costs; web app performance needs to contextually know when to schedule remote computation offloading
- Adaptive Cloud Offloading for Vehicles (Ashok2016)
  - Vehicles themselves are mobile devices, but have large sensory data inputs and extraordinary computational needs, so adaptive offload specific applications and decide when/which

Conclusion

- Tango provides a way to synergistically toggle client/edge resources by having both operate in parallel with appropriate synch-points
- Tango provides improvements to both computationally-intensive and network-intensive applications with little impact to SW devs
- Resounding focus has been to adaptively decide when to switch leadership/offloading and to apply it to other “traditionally not” mobile applications

Backup

References

1. Gordon, Mark S., David Ke Hong, Peter M. Chen, Jason Flinn, Scott Mahlke, and Zhuoqing Morley Mao. "Accelerating mobile applications through flip-flop replication." In Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services, pp. 137-150. 2015.
   Accessed at http://web.eecs.umich.edu/~zmao/Papers/Tango.pdf

2. Xu, Mengwei, Feng Qian, Mengze Zhu, Feifan Huang, Saumay Pushp, and Xuanzhe Liu. "DeepWear: Adaptive local offloading for on-wearable deep learning." *IEEE Transactions on Mobile Computing* 19, no. 2 (2019): 314-330.
   Accessed at https://arxiv.org/pdf/1712.03073.pdf

3. Jeong, Hyuk-Jin, InChang Jeong, Hyeon-Jae Lee, and Soo-Mook Moon. "Computation offloading for machine learning web apps in the edge server environment." In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 1492-1499. IEEE, 2018.
   Accessed at https://ieeexplore.ieee.org/abstract/document/8416417

4. Ashok, Ashwin, Peter Steenkiste, and Fan Bai. "Adaptive cloud offloading for vehicular applications." In 2016 IEEE Vehicular Networking Conference (VNC), pp. 1-8. IEEE, 2016.
   Accessed at https://mobile.cs.gsu.edu/aashok/Papers/14_2016_VNC.pdf