DIPLOMA: Consistent and Coherent Shared Memory over Mobile Phones

Jason Gao*, Anirudh Sivaraman*, Niket Agarwal†, HaoQi Li* and Li-Shiuan Peh*
*M.I.T. Computer Science & Artificial Intelligence Laboratory, Cambridge, MA, USA {jasongao,anirudh,haoqili,peh}@mit.edu
† Google Inc. , Mountain View, CA, USA niketa@google.com

***Abstract*—[1] Location-based services for mobile devices are pervasive, and frequently process data sensed from nearby devices as relevance is often dependent on proximity. Yet, today's services routinely use the client-server programming model which leads to sensed data being sent through the cellular network to a centralized server for processing. Harnessing the compute power of mobile devices to process data locally could ease bandwidth pressure on already overloaded cellular access networks and improve response times. Realizing this vision requires a way to easily program a collection of mobile devices connected over ad-hoc wireless. This paper presents DIstributed Programming Layer Over Mobile Agents (DIPLOMA), a programming layer and distributed shared memory system that provides coherent relaxed-consistency access to data residing on different mobile phones across a large geographic area. Our key insight is in translating the shared memory model from parallel computing to mobile computing, while addressing the unique challenges that mobility and unreliable wireless networking present in achieving consistency and coherence. We designed, prototyped and deployed DIPLOMA on 10 Android phones, evaluating it against another 10 phones running a conventional client-server setup over both 3G(HSPA) and 4G(LTE) networks. On DIPLOMA, we implemented a Panoramio-like service as an example of a popular and representative location-based service, as well as a synthetic benchmark to measure response time, cellular bandwidth consumption, and power consumption. We also simulated large scale scenarios (up to 160 nodes) on the ns-2 network simulator. Compared to a client-server setup, our system shows response time improvements of 10X over 3G and 2X over 4G. We also observe cellular bandwidth reductions of 96%, comparable energy consumption, and a 95.3% request completion rate with coherent caching.**

I. Introduction

Mobile devices are now ubiquitous. Equipped with sophisticated sensors such as GPS, camera, accelerometer and more, they already sense and generate large amounts of data. With quad-core [1] phones now on the market, smartphones will increasingly be able to compute on the sensed data in-situ as well. Yet, mobile phone applications still use the conventional client-server model, with a thin client front-end on the phone delegating compute-intensive tasks to servers in the cloud. This model is widely used for simplicity, but has several disadvantages in a mobile context:

1) **Overloading of cellular access networks:** Wireless spectrum is at a premium [2]. Next-generation cellular data networks (4G/LTE) are unlikely to fix this for two reasons: 1) 4G networks are now a substitute for home broadband; 2) Higher screen resolutions are increasing user demand for high bandwidth content such as streaming video. A recent study projected demand to exceed capacity on cellular networks by 2014 [3].
2) **Long and variable latencies:** Cellular networks are characterized by long and highly variable latencies, degrading application response times [4], [5]. Our own measurements in Section IV confirm that 3G latencies can be as high as 50 seconds. 4G latencies are currently significantly better (Section IV), but performance on 4G networks will also degrade as user adoption increases.
3) **Poor battery life:** Cellular data transmission drains energy [6], a primary resource for mobile phones.
4) **Monetary cost:** Cellular service plans are increasingly metered and monthly caps are common [2].

We propose moving to a shared memory programming model for location-based services, addressing the issues above by leveraging free, energy-efficient, and low-latency adhoc WiFi to replace cellular accesses when possible. Application developers see a single global address space as our programming layer creates a shared memory abstraction and hides the underlying mobility and phone-to-phone coordination. Thus, we make the following contributions in this paper:

1) We design and implement DIPLOMA, a Distributed Programming Layer Over Mobile Agents, enabling distributed programming by exposing a shared memory model to the application developer (Sections II and III).
2) We implement an app similar to the popular location-based photo sharing service on Google Maps, Panoramio [7], and a synthetic benchmark, and measured substantial benefits in latency and cellular bandwidth reduction compared to a conventional client-server implementation on 3G and 4G (Section IV).

II. The Design and Semantics of DIPLOMA

At a high level, a collection of mobile smartphones is a distributed system with each device having a processor core and memory. Devices are interconnected by short range radios such as ad-hoc WiFi. We propose that devices cooperate and share their memory[1] to form a distributed shared memory (DSM) system to present a familiar interface to developers. However, typical DSM systems use static nodes connected over a reliable interconnect, while a collection of smartphones represents mobile nodes connected via unreliable wireless

[1]This work is supported by the Singapore-MIT Alliance for Research and Technology Future Urban Mobility IRG, and the United States Department of Defense NDSEG fellowship.

[1]Mobile apps are typically sandboxed, so their effects on the system are isolated, mitigating security concerns. Additionally, future mobile virtualization can further isolate DIPLOMA apps [8].

networking. To address device mobility, we divide a geographical area into a 2D mesh of regions. Within each region, we abstract the collection of all phones in the region into a *single, reliable and immobile* Virtual Core (VCore) with its own memory (Section II-A). To address the unreliability of the wireless interconnect, we relax our memory consistency model (Section II-B). Additionally, we cache to speed up remote reads, and propose Snoopy, Resilient Cache Coherence (SRCC) to maintain coherence (Section II-C).

A. The Virtual Core layer (VCore)

VCores provide the abstraction of static reliable cores interconnected via a 2D mesh. We leverage Virtual Nodes(VN) [9], which abstracts a collection of unreliable mobile nodes in direct communication range of each other[2] into a stationary reliable virtual node. In the original VN system [10], a large geographical area like a city is first divided into equal-sized regions. Mobile nodes can infer their region via localization (e.g. GPS). Region size is chosen based on radio range, such that messages sent from one region can be heard by all nodes in the region, as well as in all neighboring regions. All physical nodes in a region participate in a state replication protocol to emulate a single VN per region.

The nodes elect a leader using a simple algorithm. Each node, on entering a new region, sends a leadership request to all nodes. If the leadership request is not rejected, the node claims itself as the leader and sends out regular *heartbeat* messages announcing its leadership. If a non-leader misses a certain number of *heartbeats*, it sends out a leadership request.

The client nodes broadcast requests to their local region. The leader, and non-leaders, run the same server application code. All nodes receive client requests and process them according to the application code. Only the leader node sends responses; others buffer responses until they hear the same response message from the leader. By observing the leader's replies, the non-leaders synchronize their application state to the leader and correct themselves upon a state mismatch.

The only practically deployed implementation of VN is described in [10], on a small set of PDAs. Another implementation [11] simulates VNs on the ns-2 [12] simulator. These original VN systems run into problems in practice due to unpredictable mobility and unreliable networking. Regions could become unpopulated, causing VNs to lose state. Wireless contention and range issues can create multiple leaders if nodes do not hear *heartbeats*, causing inconsistent state.

Proposed Virtual Cores. To address these problems, we propose a new implementation called Virtual Cores (VCores). A VCore is the leader in a group of mobile nodes in a single region. Most anomalies in Virtual Nodes occur when the elected new leader is out-of-sync with the old leader. VCores correct this via occasional coordination with a reliable cloud server using cellular networks like 3G (HSPA) or 4G (LTE).

Region boot-up: When the first mobile node enters a region, it broadcasts a leadership request message. If there is a VCore running here, it replies to the request and the new node becomes a non-leader. If the new node does not hear a reply within a timeout period, it contacts the cloud to nominate itself as a leader. The cloud knows if a VCore is already running in the region, and rejects the leadership request if so. Otherwise, it sends the latest shared memory state of this region back to the node, which then boots itself as the region's new VCore.

Leader (re)election: The VCore provides a stationary, reliable core abstraction until it leaves the region. At this point, it broadcasts a *LEADER_ELECT* message back to the old region. The nodes in the old region receive this message and reply with a *LEADER_NOMINATE* message. The old VCore randomly chooses one to be the new VCore and sends it a copy of the shared state with a *LEADER_CONFIRM* message. The new VCore sends a final *LEADER_CONFIRM_ACK* message to the old VCore. If the election fails due to message losses or if the old region is unpopulated, the old VCore sends the shared state to the cloud for later retrieval by a new VCore. The above steps ensure that if the region is populated, exactly one node in this region will be selected as the new VCore.

No state replication: In the original VN, the leader's state is replicated on all non-leaders, which keep their state synchronized with the leader by observing requests and the leader's replies. We eliminate replication since it does not improve reliability: the cloud server has to confirm leadership requests anyway to ensure consistent state.

B. The DIPLOMA Shared Memory layer (DSMLayer)

DSMLayer is implemented as an API that runs atop the immobile and static VCore abstraction which is overlaid over individual phones. DSMlayer glues VCores in a grid/mesh topology, communicating via wireless multi-hop messages between adjacent VCores. The phone currently running the VCore for a region contributes part of its memory towards the global shared memory, addressed through variable names rather than binary addresses. These variables make up the *shared address space* of DSMLayer. Each shared variable resides on one VCore, its *home* VCore. Variables are accessed consistently through the **Atom** primitive, which is a block of instructions executed atomically on the shared variables resident on a single *home* VCore. To execute an Atom, it is multi-hop forwarded[3] from the originating VCore to the *home* VCore and executed on its portion of shared memory.

Atoms are atomic, and always execute once or fail completely. They are equivalent to a critical section, or an *acquire-release* block in Release Consistency (RC) [13]. We discuss similarities and differences with RC in detail in Section V. We guarantee relaxed consistency [14] by default and allow Atoms to be reordered by the unreliable wireless network. To optionally enforce stricter ordering between atoms, we provide *AtomFence*, a per-*home* VCore memory fence primitive that can be executed before an Atom to guarantee that all previous Atoms occurring in program order in the thread

[2]DSMLayer, described later, removes this constraint so deployments can span arbitrarily large geographic areas.

[3]Beyond a certain threshold of hop count, ad-hoc WiFi energy and latency will exceed those of cellular networks, and a hybrid cloud/WiFi solution would be better

have completed. The use of AtomFence is optional: for some applications, allowing reordering improves performance.

Additionally, DIPLOMA provides *at-most-once* [15] execution semantics for Atoms by logging the reply when an Atom is executed. Thus, if a duplicate request is received due to a retry, the logged reply is sent back without re-execution.

C. Snoopy and Resilient Cache Coherence (SRCC)

Accesses to remotely homed data result in round-trip (possibly multi-hop) communications between the requesting and *home* VCores; resending lost messages exacerbates these delays. Caching addresses this problem, but necessitates a coherence protocol. We explain our design choices below.

Traditionally, coherence protocols are either broadcast-based [16] or directory-based [17]. In a wireless context, the latency of an extra hop (required by directory-based protocols) is high and communication is inherently broadcast, so *broadcast*-based protocols are a better fit. Further, *write update* protocols are more suitable than *write invalidate* protocols since write update protocols result in fewer messages exchanged. They consume more bandwidth by carrying the shared data in each message, but WiFi bandwidth is sufficient. Additionally, we use a *write-through, no-write-allocate* cache to ensure writes do not appear in the local cache until the local VCore receives a write update confirming the write is complete at the remote home VCore. To ensure memory consistency, all cached copies in the system must see the same **order** of reads and writes to a particular memory address. We build on timestamp snooping [18] and INSO [19], which are multiprocessor broadcast-based protocols that achieve ordering on unordered networks by assigning ordered numbers to coherence messages and presenting them in order to the destination caches. INSO and timestamp snooping rely on a highly reliable interconnect, however, making them unsuitable for wireless networks. *DIPLOMA requires a novel write update, snoopy (broadcast-based) cache coherence protocol resilient to unreliable networking.*

We design a Snoopy and Resilient Cache Coherence (SRCC) protocol. SRCC guarantees that memory operations to the same shared variable owned by any *home* VCore are seen by all remote caches in the same order. To ensure that all VCores see the *same* global order of Atoms, each *home* VCore keeps a counter called *global_order* maintained by DSMLayer. This counter indicates the number (order) that the next Atom (which may contain load/store instructions to this *home* VCore's shared variables) will be tagged with. This counter is initialized to 1. Each VCore also maintains a *local_order*, which indicates which number (order) this VCore will accept next, also initialized to 1. A VCore accepts a write update when the *global_order* of the write update equals its current *local_order*, and subsequently increments *local_order*. Write updates with higher orders are buffered until their turn arrives. Figure 1 walks through one such transaction of SRCC.

TABLE I
DIPLOMA INTERFACE METHODS.

Method	Implemented by → Called by	Invoked on	Description
long **makeAtomRequest**(long atomId, long destVCoreX, long destVCoreY, boolean isWrite, byte[] data);	DSMLayer → Programmer	Requesting region	Request to execute a predefined Atom (identified by atomId) on a destination VCore. Can include data. Returns a long to identify the request.
Atom **handleAtomRequest**(DSMLayer.Block b, Atom c);	Programmer → DSMLayer	Target region	Execute an Atom on the local portion of shared memory and return a reply Atom.
void **handleAtomReply**(Atom a);	Programmer → DSMLayer	Requesting region	Callback for receiving an Atom reply.
void **atomFence**(long destVCoreX, long destVCoreY);	DSMLayer → Programmer	Requesting region	Block until all pending Atoms have finished at the destination region.

III. DIPLOMA IMPLEMENTATION

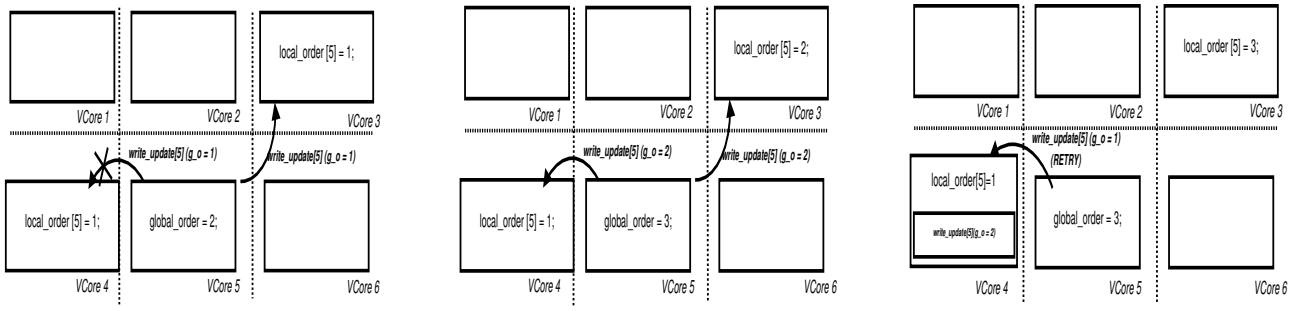
A. DIPLOMA's API

Table I lists the DIPLOMA API. First, the application programmer wishing to use DIPLOMA implements the UserApp *i.e.* the service to be provided in the network. Within the UserApp, the programmer implements the function bodies of the Atoms that can be executed on any specified *home* VCore at run time. Atoms can contain arbitrary Java code that may contain reads and writes on multiple variables on one home VCore. The application logic in the UserApp requests the execution of an Atom by calling a method exposed by DSMLayer, `makeAtomRequest`. Behind the scenes, the DSMLayer routes the request to the specified home VCore, where `handleAtomRequest` is invoked with a reference to the local portion of shared memory on which to execute the Atom. `handleAtomRequest` (implemented by the programmer) returns a reply which is routed back to the originating VCore and passed to `handleAtomReply` (also implemented by the programmer). The programmer may also call `atomFence` to block program execution until all pending and in-flight Atom requests to a home VCore from a requesting VCore have either succeeded or failed / timed-out.

B. Prototype Design

We implemented DIPLOMA as an Android application running on Nexus S phones with 3G and Galaxy Note phones with 3G and 4G. Our implementation is comprised of 3 components: the application-developer-implemented app (UserApp), which runs on top of the DIPLOMA Shared Memory Layer (DSMLayer) with caching (SRCC) (enabled optionally), which runs on top of the Virtual Cores layer (VCore). All 3 components run in a single thread to eliminate inter-thread communication. This also ensures execution of Atoms cannot be interrupted by VCore protocol messages. Atoms are also marked with Java's `synchronized` keyword to disallow concurrent access.

A second thread runs a busy-wait loop to receive packets on the adhoc WiFi interface. To communicate between the first and second threads, a Mux is implemented in a third thread, so packets can always be en/dequeued regardless of activity in the first thread. When a VCore needs to upload shared memory to the cloud server, the VCore layer pauses the DSMLayer, serializes the shared memory to JavaScript Object Notation (JSON), and sends it over the cellular network to the server.



(a) State of system when first write update of VCore 5 is broadcast. The update does not reach VCore 4.

(b) State of system when second write update of VCore 5 is broadcast. This update reaches both VCores 4 and 3.

(c) First write update of VCore 5 retried. Second write update is buffered at VCore 4 with a *global_order* of 2.

Fig. 1. Walkthrough example of SRCC for two writes to VCore 5. Only VCores 3, 4 ,5 are detailed for clarity.

C. Practical Considerations

Next, we discuss some of the issues that arise in a practical deployment of DIPLOMA, describe how our implementation deals with them and continues to operate correctly.

Wireless range more limited than assumed. DIPLOMA's default behavior for VCore assumes that the exiting leader remains in wireless range of its old region when it moves to a neighboring region, so that it can elect a new leader. If the old leader moves out of range before electing a new one, it sends its state to the cloud server so that a new node may download and boot the VCore later. If the wireless range turns out to be much smaller than expected, it could cause many region reboots, hurting latency and completion rate. Our benchmark deployment (Subsection IV-A) shows that WiFi wireless range is sufficient: 57% of leader hand-offs succeed without requiring a region reboot, enough to achieve completion rates up to 95.3%.

Resilience to node failures. DIPLOMA monitors for low battery or user opt-out, and initiates leadership hand-off. It also monitors for unexpected node failures with a leader-to-cloud heartbeat (every 120 seconds in our implementation), so that the server will become aware of node failures and allow a new node to become the leader with the last known state.

Atomic execution of Atoms in the face of interrupts. In our implementation, the DSMLayer runs in the same thread as the VCore layer and message handling methods are marked with Java's `synchronized` keyword to ensure that VCore protocol messages cannot interrupt Atom execution. Additionally, when the VCore layer hands off leadership, it pauses the DIPLOMA layer, ensuring that no DIPLOMA Atom requests are processed by the old VCore while or after the new VCore receives the state. Instead, any DIPLOMA Atom requests received during the hand-off a dropped and resent to the new VCore later by the requesting VCore.

Intermittent cellular connectivity. When a node needs to make a cellular access, e.g. upon entering an empty region, it sends a request to the server to become the VCore, retrying if the server is unreachable. Thus, for DIPLOMA to work, the cellular connection must be eventually available. Current metropolitan cellular networks exhibit this behavior; in our benchmark deployment (Subsection IV-A), 3G was available 98% of the time.

IV. Evaluating DIPLOMA

We implemented two mobile applications to evaluate DIPLOMA vs cloud-only solutions: a synthetic benchmark that is scripted to generate a specified percentage of read and write requests to a random VCore, and a Panoramio-like [7] app. For comparison, we also implemented cloud-only applications functionally equivalent to the DIPLOMA versions, but relying purely on HTTP requests over 3G/4G to a single-threaded Python web server. The server ensures that accesses to the shared memory are consistent, and provides the same functionality. The server is located in the same geographic region as the phones to minimize backbone Internet latency.

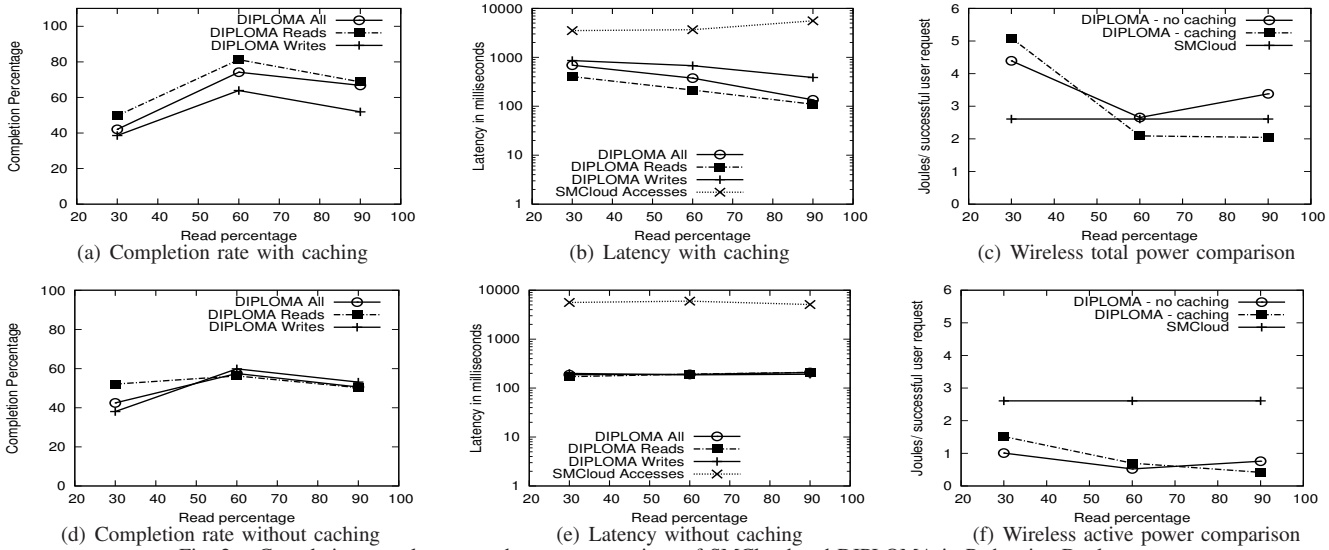
A. Benchmark App

We carried out a deployment with our synthetic benchmark running on Google Nexus phones with 3G radios in a covered pavilion last year. The area is divided into four regions of 5mx5m per region. Ten volunteers held two phones each, with DIPLOMA running on one phone and cloud-only shared memory (SMCloud) on the other. The volunteers walked among the regions with the phones and indicated which region they were in at a given time. We evaluated DIPLOMA under combinations of SRCC caching disabled/enabled and varying read/write distributions. We measured DIPLOMA's performance against the cloud-only version (SMCloud) using: (1) average latency of successful requests, (2) completion rate of requests, (3) average energy consumed per successful request, and (4) cellular data consumption. Our methodology and results are detailed below.

Average latency: User interface interactions are timestamped to obtain end-to-end request latencies. We compare DIPLOMA to SMCloud in Figures 2(b) and 2(e)[4]. Request latencies for DIPLOMA are typically an *order of magnitude lower* than those in SMCloud.

Without caching, read and write latencies do not vary greatly across read vs. write distributions, as they both incur hops to remote HOME VCores. With caching enabled, high read percentages (90%) show significantly decreased latencies: when requests are serviced at the local VCore from its cache,

[4]SMCloud results appear in both the cache and no cache trials because we ran it in every trial simultaneously against DIPLOMA to control for cellular conditions between trials.



(a) Completion rate with caching (b) Latency with caching (c) Wireless total power comparison

(d) Completion rate without caching (e) Latency without caching (f) Wireless active power comparison

Fig. 2. Completion rate, latency and power comparison of SMCloud and DIPLOMA in Pedestrian Deployment

hops to remote VCores can be eliminated. Write latencies are significantly higher than read latencies because they require write updates to be broadcast to the entire system. This increased write latency is even more pronounced at lower read (higher write) percentages (60%, 30%) as the write updates increase network congestion, and even impact and increase read latencies, too. Thus, caching is advantageous in applications with a higher proportion of requests being reads.

Request completion rate: We calculate the percentage of issued requests that complete (Figures 2(a) and 2(d)). Again, we measure reads and writes separately and in aggregate, and compare the completion rate of DIPLOMA to SMCloud.

Without caching, the completion rate of application-level requests is 57%, and does not vary between read/write distributions, as expected. With caching, at 60% reads, *80% of application-level requests on DIPLOMA complete*. Note that these application-level requests incur an extra wireless hop from a client app to the UserApp on the region's VCore, which may fail before DIPLOMA is even invoked; the completion rate of the DIPLOMA Atoms alone is 90.9% for 60% reads, and *95.3% for 90% reads.* Caching allows many read requests to be successfully serviced from the local VCore even when a read request to the remote VCore fails.

The completion rate is lower at lower read distributions (30%) due to several factors: more requests are writes, which have lower completion rates than reads because they cannot be cached and must be sent to remote regions; higher wireless contention due to more write updates being broadcast to the entire network, resulting in dropped application packets. This is seen in the disparity between DIPLOMA-level and application-level request completion rates. The application-level implementation does not implement a retry/ack mechanism, unlike DIPLOMA. Thus, at 90% reads, though 95.3% of the DIPLOMA Atoms successfully complete at the VCore, the local VCore's subsequent reply to the client node is only received in 66.8% of requests.

In contrast to DIPLOMA, in SMCloud we observe a 100% completion rate (not shown in figure) of requests, but requests can take as long as 55 seconds to complete in our evaluations. Such high latencies are instances of a problem called Bufferbloat [20]. We discuss DIPLOMA's completion rate further in Section IV-C.

Power consumption: We use the Monsoon power meter [21] to build an energy model for the Nexus S devices. Devices running DIPLOMA use adhoc WiFi, so energy for access point scanning and associations is not incurred. Consistent with previous studies [6], [22], our results shows that the energy of a WiFi transmission is significantly less than that of 3G. In our applications, a single HTTP request over 3G is measured to consume 2.6 Joules, while a single WiFi packet transmission might consume only 0.066 J. We do not factor into account energy expended in localisation because this is a task common to both SMCould and DIPLOMA.

We create a linear regression for receive and transmit energy across several packet sizes (1k, 2k, 4k, and 8k bytes) (R-squared=0.999 for Tx, 0.959 for Rx). This regression is applied to average packet sizes calculated from the deployment logs to obtain per-packet energies for each of the deployment trials, obtaining total energy consumed by WiFi and 3G in each trial.

WiFi idle power (turned on, but not receiving or transmitting) is also measured, and then calculated for each of the trials using experimental run time. Again, consistent with [6], [22], we find that WiFi consumes significant idle power: with only the 3G radio turned on, current consumption is 149 mA. Once adhoc WiFi is turned on, current consumption increases 46% to 218 mA, without any WiFi traffic.

We use these observations to measure the power consumption of both DIPLOMA and SMCloud by processing logs offline. Both SMCloud and DIPLOMA applications wait for 2 seconds between requests[5]. The 3G radio does not return to a low power state between requests in SMCloud due to cloud

[5]2 seconds being a realistic time between user interactions. We choose not to batch requests since they are user-initiated, and to maintain a responsive user experience, should not be delayed

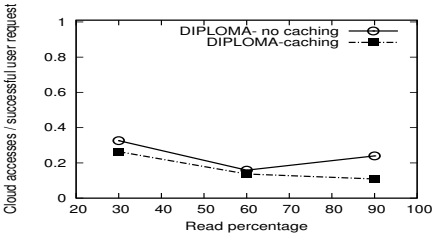


Fig. 3. Number of cloud accesses

accesses being much more frequent; therefore, measurements include 3G tail energy [6] for all cloud accesses. Taken together, these measurements give total energy consumed by WiFi + 3G for DIPLOMA, and total energy consumed by 3G for SMCloud, per trial. These totals are then divided by the number of successful requests per trial to arrive at an average energy consumed per successful request per trial.

As we see in Figure 2(f), DIPLOMA *reduces active wireless energy consumption by up to 94%* per successful request. However, when WiFi idle power is factored in, DIPLOMA is more energy efficient only with caching enabled at 60% read distributions or higher (Figure 2(c)), due to WiFi idle power being quite significant. This highlights the need for better power management of WiFi radios when used in adhoc mode for short-range phone-to-phone communications.

Cellular access reduction: SMCloud solely communicates with the cloud server over the cellular data network, so a cloud access is incurred for every read or write request to shared memory. In contrast, DIPLOMA incurs cloud accesses only for region bootups and leadership changes, which occur due to mobility rather than application interactions, so these accesses are amortized over the requests from the application. Hence, we divide the total number of successful cloud accesses by the number of successful requests (DIPLOMA was able to reach the cloud through 3G in 98% of attempts). These results are shown in Figure 3 where the x-axis represents the percentage of reads in our benchmark app.

DIPLOMA without caching averages 0.21 cloud accesses per successful request, a 79% reduction from SMCloud, and DIPLOMA with caching averages 0.14 cloud accesses per successful request, a 96% reduction. Caching leads to more successful requests and quicker responses, while the number of cloud accesses remains the same. This advantage is more pronounced at higher read percentages.

B. Panoramio-like App

We implemented a Panoramio-like app on Galaxy Note phones to demonstrate that popular consumer mobile apps today can be readily ported onto DIPLOMA. In the app, we use the shared memory abstraction provided by DIPLOMA to retrieve and update photo data. Users (clients) can *take* pictures of interesting things where they are, and they can also *get* pictures taken by other users. The photos are stored in the same region that they are taken in. If a user desires to view photos from a remote region, *gets* can traverse multiple hops on their way to a remote region. The phones serve double duty by both participating in DIPLOMA (as leaders or non-leaders)

TABLE II
PANORAMIO-LIKE APP LATENCIES OVER 3G

	takes CameraSM	*takes* CCloud	*gets* CameraSM	*gets* CCloud
mean	144 ms	2558 ms	217 ms	2279 ms
median	109 ms	2465 ms	161 ms	2229 ms

TABLE III
PANORAMIO-LIKE APP LATENCIES OVER 4G

	takes CameraSM	*takes* CCloud	*gets* CameraSM	*gets* CCloud
mean	144 ms	546 ms	178 ms	469 ms
median	107 ms	534 ms	159 ms	469 ms

and being the clients of the application themselves. To reduce the size of data transfers, we apply JPEG compression to all pictures before transmission. We also implement a functionally equivalent cloud version (CCloud) of the same app (accessed through 3G/4G) and compare the DIPLOMA version without caching (CameraSM) to the cloud based version in terms of completion rate and request latencies.

We carried out a deployment of Panoramio on 20 Galaxy Note phones over 3G and 4G networks this year, with 10 phones running CameraSM, and another 10 running CCloud. Phones are placed statically and uniformly across 6 regions (5mx5m each) within an open indoor space. Two people walk around the phones clicking on buttons simultaneously on CameraSM and CCloud pairs of phones, taking and getting pictures. We present mean and median latencies in Tables II and III, omitting distributions for brevity. Similar to the benchmark application, we also measured the number of cloud accesses per application-level *get* or *take* request for both CameraSM and CCloud. Since CCloud makes a cloud access on every request, this number is 1 for CCloud on both 3G and 4G networks. For CameraSM, we observed 0.29 cloud accesses per request on 3G, and 0.22 accesses per request on 4G . Since the phones were static, these accesses were primarily due to leader-to-cloud heartbeats which occurred at 2 minute intervals. The heartbeat interval allows us to trade off between number of cloud accesses and the reboot time of an unpopulated region. We observed a high completion rate of 98.6% for CameraSM across 573 requests, and 100% for CCloud across 564 requests. These results show DIPLOMA outperforming both 4G and 3G cloud implementations in response times while retaining high completion rates.

As Panoramio has substantial write traffic, our write update caching protocol leads to excessive WiFi traffic (approximately 6KB write updates for every region when a picture is taken, plus associated ACKs) and was turned off in this deployment. In hindsight, applications like Panoramio would work better with a write-back protocol. We don't have power comparisons for Panoramio as 4G has more sophisticated power management, making it difficult to apply a power model naively to our activity traces to get accurate power estimates.

We also conducted outdoor mobile deployments with this app, but saw high loss rates over ad-hoc Wifi, which could be due to the large packet size of images, high WiFi interference in the area, and/or poor antennas on the Notes. We are in the process of diving further into these ad-hoc WiFi problems and investigating potential optimizations.

C. Simulation studies

We use ns-2.37 [12], a discrete event network simulator, to evaluate our system at scale with the synthetic benchmark. Node mobility is simulated with the Random Way Point model with three settings: slow, medium and fast (Figure 6). Node movements are constrained to a $350m \times 350m$ terrain and the radio range is fixed at 250m. 250m is well within the transmission range of 802.11p or DSRC [23], which we expect will become the basis for adhoc communications for distributed mobile apps. This radio range dictates our region size since every broadcast has to be heard by the neighboring regions as well, resulting in 4 x 4 regions. Since we have 4 regions in each dimension, we also evaluate the efficiency of caching for requests that traverse between 0 and 3 hops. Each simulation lasts 40000 seconds.

Variation of node density. We vary the number of nodes from 40 to 160 to study the effect of increasing node density on DIPLOMA's performance. The resulting node density is close to typical car densities in US cities which vary from 1700-8000 cars per square mile [24], or about 80-380 cars for our $350m \times 350m$ terrain. Figure 4 shows the effect of varying the number of nodes on the completion rate of DIPLOMA. We see that increasing the node density significantly improves the performance of DIPLOMA. Also, after a threshold density of 80 nodes, the completion rate saturates near 100%.

Usefulness of caching. One intuitively expects caching to be more useful for reads to farther away regions. Writes would also take longer since they trigger updates in SRCC. To study this, in Figure 5 we plot the completion time of a request with caching enabled for varying node speeds. The numbers are normalized to a no-caching implementation. The proportion of reads and writes is kept equal to avoid any bias. We see that caching improves latency for all requests spanning 1 hop or more. On average, the 1-hop, 2-hop and 3-hop requests have a 35%, 45% and 48% lower request latency as a result of caching. However, the incremental benefit of caching decreases with increasing hops. This is understandable since write latencies scale linearly with hop count.

In summary, our simulation results demonstrate the effectiveness of caching and show how penetration of DIPLOMA affects performance. We envision that a large city scale deployment will have sufficient density to achieve a completion rate close to 1, while simultaneously providing the latency and cellular utilization benefits we observed in our deployments.

V. Related Work

DIPLOMA is related to several systems in Computer Architecture, Sensor Networks, Distributed Algorithms and Distributed Systems. We outline key similarities and differences.

Computer Architecture: Most commercial architectures, such as x86 [25] and IBM PC [26], stay close to sequential consistency [27] by reordering only certain instruction combinations. Similar to DIPLOMA, some processor architectures (Alpha [28], Sparc [29]) aggressively reorder all instructions by default and provide memory fences for the programmer or compiler to enforce ordering if required.

Among research systems, DIPLOMA is closest to Release Consistency (RC) [13]. RC defines memory operations as either *ordinary* or *special*. *Special* operations are either synchronization or non-synchronization accesses. Synchronizing accesses are either *acquire*s or *release*s. Memory accesses within an *acquire-release* block form a critical section and execute atomically, provided each critical section is protected with enough *acquires*. Every **Atom** in DIPLOMA implicitly begins with an *acquire* and ends with a *release*, guaranteeing exclusive access to the Atom's shared variables.

DIPLOMA has similarities to Transactional Memory [30]: Atoms are like transactions, but transactions allow atomic modifications to arbitrary portions of the memory, while Atoms operate on memory belonging to one VCore alone.

Sensor Networks. Several programming languages have been proposed for collections of resource-constrained devices. Kairos [31], an extension of Python, abstracts a sensor network as a collection of nodes which can be tasked simultaneously within a single program. Pleiades [32] borrows concepts from Kairos and adds consistency support to the language. These proposals are tailored to static sensor nets and do not deal adequately with mobility.

Distributed Algorithms. Most distributed algorithms for mobile agents tackle programmability by first emulating a static overlay. Virtual Nodes (VN) [9] is one such abstraction. Section II-A discussed the practical issues with VN. Geoquorums [33] provides consistency support using a quorum-based algorithm to construct consistent atomic memory over VNs, but it assumes reliable physical layer communication. [34] presents complex algorithms to implement reliable VNs over an unreliable physical network through consensus, which is expensive in practice on wireless networks.

Distributed Systems. There are several loosely coupled distributed systems that explore varying notions of consistency. Bayou [35] allows eventual consistency between data copies residing on differing replicas, which could be mobile nodes or dedicated servers. All replicas are equal and merged opportunistically using an anti-entropy protocol. In contrast, DIPLOMA maintains one authoritative copy of the data (the VCore) and actively resolves conflicts using cache coherence. CODA [36], is a file system for mobile devices with unreliable cellular connections. DIPLOMA instead targets shared memory and assumes modern cellular connections are far more reliable (albeit with very long and variable latencies). InterWeave [37] is a hierarchical consistency model with varying consistency guarantees for different levels ranging from hardware shared memory to weakly consistent shared memory across the Internet. It is significantly different from our system since DIPLOMA is homogeneous and flat and operates primarily on wireless LAN links. Semantically, TreadMarks [38] is the closest to DIPLOMA since it implements release consistency. Further, similar to DIPLOMA, it implements Distributed Shared Memory. However, TreadMarks is tailored to a workstation environment with highly reliable LAN links. Mobility and network unreliability are new problems DIPLOMA tackles.

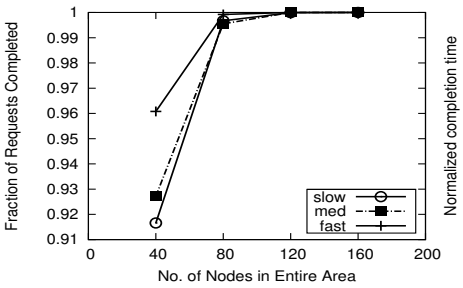


Fig. 4. Completion rate w/o caching

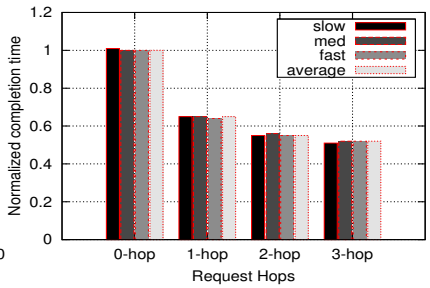


Fig. 5. Request completion latency

Parameter	slow	med	fast
Min. speed (m/s)	0.73	1.46	2.92
Max. speed (m/s)	2.92	5.84	11.68
Min. pause time (s)	400	200	100
Max. pause time (s)	4000	2000	1000
Mean cross time (s)	48	24	12

Fig. 6. Simulation settings

VI. Conclusion

In this paper, we showed that shared memory, a programming paradigm that is widely adopted for parallel programming, can be realized on mobile devices. As mobile applications for public services such as transportation become increasingly pervasive, we envision the opportunity to piggyback systems software such as DIPLOMA onto large numbers of mobile devices, realizing a powerful mobile computing platform that can offload communications from cellular networks and computation from servers. This paper takes a step towards this vision, investigating shared memory as an alternative programming model to client-server.

VII. Acknowledgements

We thank Prof. Niraj Jha of Princeton for helpful discussions on cache coherence, Prof. Nancy Lynch of MIT and Prof. Seth Gilbert of NUS for introducing us to Virtual Nodes, Jiang Wu of CUNY for sharing his ns2 implementation of Virtual Nodes, and the many volunteers who participated in our experiments.

References

[1] "Samsung Galaxy Note S3," www.samsung.com/global/business/data/Exynos420QUAD.pdf.
[2] "Verizon Killing Unlimited Data Plans Soon," http://huffingtonpost.com/2011/06/21/verizon-unlimited-data-plan_n_881091.html.
[3] "Mobile Broadband Capacity Constraints And the Need for Optimization," http://www.rysavy.com/Articles/2010_02_Rysavy_Mobile_Broadband_Capacity_Constraints.pdf.
[4] E. Koukoumidis *et al.*, "Pocket cloudlets," in *Proceedings of ASPLOS XVI*. New York, NY, USA: ACM, 2011, pp. 171–184.
[5] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, pp. 14–23, 2009.
[6] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications," in *Proc. ACM IMC*, Nov. 2009.
[7] "Panoramio," http://www.panoramio.com/.
[8] J. Andrus *et al.*, "Cells: a virtual mobile smartphone architecture," in *Proceedings of the Twenty-Third ACM SOSP*. New York, NY, USA: ACM, 2011.
[9] S. Dolev *et al.*, "Brief announcement: Virtual stationary automata for mobile networks," in *Proc. ACM PODC*, 2005.
[10] M. Brown *et al.*, "The virtual node layer: A programming abstraction for wireless sensor networks," in *Proc. Int. Wkshp. Wireless Sensor Network Architecture*, 2007.
[11] J. Wu *et al.*, "Simulating fixed virtual nodes for adapting wireline protocols to MANET," in *Proc. IEEE NCA*, 2009.
[12] "The network simulator - ns-2," http://www.isi.edu/nsnam/ns/.
[13] K. Gharachorloo *et al.*, "Memory consistency and event ordering in scalable shared-memory multiprocessors," in *Proc. ISCA*, 1990.
[14] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: A tutorial," *Computer*, vol. 29, pp. 66–76, December 1996.
[15] A. D. Birrell and B. J. Nelson, "Implementing remote procedure calls," *SIGOPS Oper. Syst. Rev.*, vol. 17, no. 5, p. 3, 1983.
[16] J. R. Goodman, "Using cache memory to reduce processor-memory traffic," in *Proc. ISCA*, 1983, pp. 124–131.
[17] D. Lenoski *et al.*, "The directory-based cache coherence protocol for the dash multiprocessor," in *Proc. ISCA*, 1990.
[18] M. M. K. Martin *et al.*, "Timestamp snooping: an approach for extending smps," in *Proc. ASPLOS IX*. New York, NY, USA: ACM, 2000, pp. 25–36.
[19] N. Agarwal, L.-S. Peh, and N. K. Jha, "In-network snoop ordering (INSO): Snoopy coherence on unordered interconnects," in *Proc. HPCA*, Feb. 2009.
[20] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Queue*, vol. 9, no. 11, pp. 40:40–40:54, Nov. 2011.
[21] "Monsoon Power Monitor," http://www.msoon.com/LabEquipment/PowerMonitor/.
[22] A. Rahmati and L. Zhong, "Context-for-wireless: Context-sensitive energy-efficient wireless data transfer," in *Proc. Mobisys*, 2007.
[23] "IEEE 802.11p," http://standards.ieee.org/findstds/standard/802.11p-2010.html.
[24] "A Free Parking Space Grows in Manhattan," http://ti.org/antiplanner/?p=3565.
[25] P. Sewell *et al.*, "x86-tso: a rigorous and usable programmer's model for x86 multiprocessors," *Commun. ACM*, vol. 53, July 2010.
[26] "IBM System/370 Principles of Operation,IBM, May 1983. Publication Number GA22-7000-9, File Number S370-01."
[27] L. Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE Trans. Comput.*, vol. 28, pp. 690–691, September 1979.
[28] "Richard L Sites, editor. Alpha Architecture Reference Manual. Digital Press, 1992."
[29] "The SPARC Architecture Manual, Prentice Hall, 1994. SPARC International,Version 9."
[30] M. Herlihy and J. E. B. Moss, "Transactional memory: architectural support for lock-free data structures," in *Proc. ISCA*. New York, NY, USA: ACM, 1993, pp. 289–300.
[31] R. Gummadi *et al.*, "Kairos: A macro-programming system for wireless sensor networks," in *Proc. SOSP*, 2005.
[32] N. Kothari *et al.*, "Reliable and efficient programming abstractions for wireless sensor networks," in *Proc. ACM SIGPLAN PLDI*, 2007.
[33] S. Dolev *et al.*, "Geoquorums: Implementing atomic memory in mobile ad hoc networks," *Distrib. Comput.*, vol. 18, no. 2, pp. 125–155, 2005.
[34] G. Chockler, S. Gilbert, and N. Lynch, "Virtual infrastructure for collision-prone wireless networks," in *Proc. ACM PODC*, 2008.
[35] D. B. Terry *et al.*, "Managing update conflicts in bayou, a weakly connected replicated storage system," in *Proceedings of the 15th ACM SOSP*, 1995.
[36] J. J. Kistler and M. Satyanarayanan, "Disconnected operation in the coda file system," *ACM Trans. Comput. Syst.*, vol. 10, February 1992.
[37] D. Chen *et al.*, "Interweave: A middleware system for distributed shared state," in *5th International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, 2000.
[38] C. Amza *et al.*, "Treadmarks: Shared memory computing on networks of workstations," *Computer*, vol. 29, February 1996.