AXaaS (Acceleration as a Service): Can the Telecom Service Provider Rent a Cloudlet ?

Nathaniel Powers, Tolga Soyata
Dept. of Electrical and Computer Engineering, University of Rochester, Rochester, NY 14627
npowers@u.rochester.edu,tolga.soyata@rochester.edu

***Abstract*—As the capabilities of smartphones expand, so do consumers' expectations for resource-intensive mobile applications. However, mobile devices are ill-suited to execute most these applications due to their hardware limitations. Computational offloading offers a way to augment mobile computation power, but it introduces a communication latency, potentially weakening or negating its advantages. To meet the user demand for high performance, we propose a new service architecture called *Acceleration as a Service* (AXaaS). We formulate AXaaS based on the observation that most resource-intensive applications, such as real-time face-recognition and augmented reality, have similar resource-demand characteristics: a vast majority of the program execution time is spent on a limited set of library calls, such as Generalized Matrix-Multiply operations (GEMM), or FFT. Our AXaaS model suggests accelerating *only* these operations by the Telecom Service Providers (TSP). We envision the TSP offering this service through a *monthly computational service charge*, much like their existing *monthly bandwidth charge*. We demonstrate the technological and business feasibility of AXaaS on a proof-of-concept real-time face recognition application.**

I. Introduction

Mobile-cloud offloading of data storage and processing to the cloud allows a mobile device to appear more powerful than it really is [1]–[7]. Although intensive processes aren't actually being handled by the device, the requisite simplicity of the user interface effectively renders the offloading routines as transparent, leading to increasing expectations for performance by consumers, who are by now, quite used to experiencing steady improvements as the norm. To enable an emerging class of resource-intensive mobile applications such as real-time face recognition, linguistic processing/translation and augmented reality [8]–[12], solutions based on an intermediary *cloudlet* device have been proposed to accelerate the computation partially at the source before it reaches the cloud [8], [13]–[15]. These techniques rely on an expensive cloudlet that amassses substantial computational power. Even if a mobile user makes an investment in such a cloudlet, continuous upgrades will be necessary to keep up with the increasing computational demand from evolving applications.

There is no service archetype in place which offers a standardized acceleration of computation for a large number of users. We introduce the AXaaS service model which can potentially connect millions of mobile device users to superior computational resources through the low latency links provided by Telecom Service Providers (TSPs). In our model, TSPs can be thought of as *renting a cloudlet* to their subscribers [16]. This cloudlet can be orders-of-magnitude faster than one that their users can purchase. Since our target applications demand the use of this cloudlet in a burst form, the accumulated usage is actually very low. This provides an opportunity for the TSPs to aggregate these burst requests from millions of users and service them in powerful datacenters, enabling the aforementioned new class of mobile applications that are currently impossible or impractical to develop. We propose Telecom Service Providers (TSPs) as the most logical provider of AXaaS, due to their existing relationship with their users. By charging a monthly fee (e.g., TFLOPs/month), TSPs may use this revenue to offset the cost of renting computational resources from cloud operators [17]–[19]. This model absolves the TSP of the burden of building and maintaining proprietary data centers. The central impetus is enabling universal access to superior computational resources in order to facilitate innovations in the development of mobile applications.

The rest of this paper is organized as follows: We provide background on existing aaS models and the face recognition algorithm and motivate AXaaS in Section II. In Section III, we make a distinctive definition of *acceleration* and consider the types of constituting computations. We introduce our AXaaS model in Section IV. An example of the AXaaS model is examined along with an analysis of TSP costs and return on investment (ROI) in Section V. Implications to end-users are given in Section VI. We explore performance evaluations in Section VII, and make our conclusions in Section VIII.

II. Background and Motivation

"As a service" (aaS) offerings provide convenient, configurable solutions to computational problems by allocating resources over the internet. To run an exciting new breed of resource-intensive mobile applications such as Real-Time Face Recognition [8], Augmented Reality, Real-time Language Translation, and Surveillance, it is possible for a user to rent *cloud instances*. However, the steps required for such a process will discourage even the most willing user. In this section, we will introduce a framework which will enable these applications with nearly zero burden to the user.

A. Distinguishing AXaaS from other aaS Services

Our profiling shows that, resource-intensive mobile applications spend a majority of their time executing a limited set of APIs, such as Generalized Matrix-Matrix Multiplication and Fast Fourier Transform, which can be *accelerated* through

978-1-4673-9501-4/15/$31.00 ©2015 IEEE

hardware accelerators [20], [21], thereby significantly improving their performance. These applications need is *burst access* to extremely high-intensity computation, rather than *continuous* access to steady, generalized computation. We observe a lack of cloud service offerings for providing generalized burst raw computation to improve the performance of these mobile applications. In this vein, we propose a new service model which shifts its focus from the *application* to the *fundamental APIs*, which are very common to many resource-intensive applications. In analyzing the AXaaS model's business viability, we will show that, the acceleration that is offered as a service can dramatically improve *user experience*, motivating the users to pay for such a new service.

B. Motivation for Acceleration as a Service

Let us assume that, a user is running a mobile-based Face Recognition (FR) application on a state-of-the-art mobile phone or tablet over a 4G cellular network. We are particularly interested in performance differences owed to computational offloading. Our assumptions on application parameters are: An image size of 145 KB, result data size of 8 KB for the set of recognized faces, a database size of 5000 faces to be recognized from, Apple iPhone 5S based on the A7 Cyclone with PowerVR G6430 GPU [22], and Nvidia SHIELD tablet, based on the Tegra K1 [23] as the mobile platforms running this application, a cloud compute capability of 500 TFLOP/s and WAN speed of 10Mbps (4G Network).

Scenario 1: Assuming that the entirety of the FR application and its data are hosted and executed on the iPhone, our experiments allow us to estimate a single face to be processed and recognized by the iphone in approximately 4.6 seconds, and by the Nvidia SHIELD in approximately 0.98 seconds.

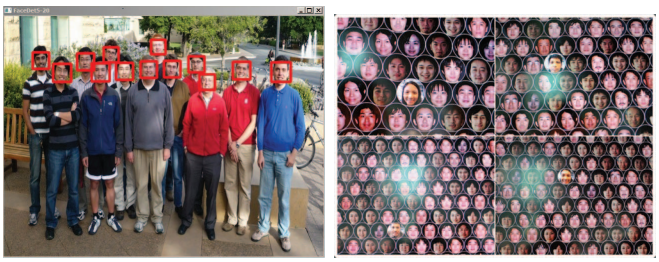

Fig. 1: Examples picture frames with 13 faces [8] (left) and 80 faces (right). File sizes are 145KB and 1.29MB, respectively.

Scenario 2: The tablet uploads the captured 145 KB image containing 13 faces shown in Fig. 1 (left) through a 4G network to some acceleration instance. This instance would rapidly accelerate FR functions and return the result to the mobile back over the 4G. While this approach introduces a communication delay of 249.2 ms, the resultant query time would be only 249.59 ms. Merely 390 μs of rapid computation in the cloud is required to produce a complete set of results.

Scenario 3: If we assume a futuristic 1Gbps network speed (e.g., 5G, which should be offered within a few years [24]), the response time to recognize the 80 faces in a single frame shown in Fig. 1 (right) would be reduced to 22.84 ms total (20.44 ms is the communication cost).

C. Quantifying the User Experience: The γ

Our key motivation is to formulate a model for AXaaS around a quantitative value for *user experience*. Clearly, users will only pay for services that either enhance their business, daily lives, or for entertainment. We observe that, it is relatively easy to define a quantitative metric for user experience:

$$\gamma = \kappa \times \lambda \tag{1}$$

where κ is the average faces per frame (*frame density*) and λ is the average frames per second (*temporal density*). Their product γ (*total faces recognized per second*) is clearly directly related to user experience, but what range of γ values correspond to what type of user experiences ?

In Scenario 1, when a smartphone or tablet solely recognizes faces, γ=0.22 and γ=1.02. In Scenarios 2 and 3, a significantly improved γ =50–3500 is achieved. These simple examples show that, while an advanced mobile device that is available today struggles to recognize even a single face within a second, by augmenting the FR application using AXaaS, the perceived value of the application can be drastically improved, thereby prompting the user to pay for such a service. While most users will happily call γ=10 a **Real-Time Face Recognition**, γ=10–100 will allow the application to be significantly more versatile, allowing the user to run it on highly populated scenes. Finally, γ =100–1000 and beyond will allow **Real-Time Surveillance** over very populated scenes, such as airports and public speeches of government officials.

D. Face Recognition Algorithm

We choose *Real-Time Mobile-Cloud Face Recognition* as our proof-of-concept application, which possesses many representative characteristics of applications that are amenable to AXaaS-based acceleration. The FR algorithm [25] can be divided into three distinct phases: **Face Detection**, which separates and extracts the actual faces from the source frame, **Projection**, which converts each detected face into a set of coefficients, and **Search**, which compares the projection of the detected face with projections of images stored in a database and returns a result with the highest degree of similarity.

Detection: To execute detection we have employed the Viola-Jones object detection framework [26], which provides competitive real-time detection rates. This portion of the algorithm is well studied [27] and highly parallelizable. Most modern smartphones have hardware accelerators for this very common function. Today's operating systems, such as Android 4.x, Ice Cream Sandwich or Jelly Bean [28] have built-in API functions for hardware acceleration for $\leq$2 faces.

Projection: is the most computationally intensive component of FR. The Eigenfaces method [29] in the OpenCV *FaceRecognizer* API [30] uses **GEMM** functions [21] and calls for two nearly identical but separable components which are necessary for FR. The first is an off-line initialization which trains a multi-dimensional eigenspace from database (DB) images. Using Principle Component Analysis (PCA) [31], a set of near-orthogonal basis functions called

eigenvectors are formed which allow each face in the DB to be represented as a unique linear combination thereof.

Database Search: The final step compares each test face projection to the collection of projections stored in the DB compiled during initialization. The prediction result corresponds to the projection in the DB with the minimum Eeuclidean distance to that of the test face.

III. Computational Acceleration (AX)

By *acceleration* (*AX*), we mean executing fundamental APIs (e.g., FFT, BLAS) 100–1000× faster (i.e., *accelerating* them) to significantly improve a user's application experience.

A. Accelerating Face Recognition

Without loss of generality, we will focus on real-time face recognition (FR) as a candidate AX application, although any computationally intensive real-time application [32]) is an excellent candidate for AX (e.g., private real-time health monitoring [33]–[36]). We have shown in Section II-D that **Projection** would particularly benefit from AX, due to the vast number of calls to the GEMM subroutines. This is a highly generalizable operation that may be used by an incredibly broad array of applications. There exist other operations at the same abstraction layer which could be used similarly, such as Fast Fourier Transforms (FFTs), sorting and sparse matrix operations. Our key idea in this paper is to accelerate only such generalized common *API functions*, rather than the *application* itself, thereby eliminating the necessity to rewrite acceleration routines for every new application.

In Section II-C, we introduced a metric γ to express the user-apparent performance of our FR application (i.e., *user experience*). γ is the total faces recognized per second, including every frame within that second. We also identified communication and computation as the pivotal deterministic factors for achieving values for γ large enough for the application to be considered useful to users. Providing acceleration (AX) for FR would undoubtedly require the most computationally intensive cloud resources, as well as low-latency communication pipelines between the user and the cloud. Each of these factors are inherently finite in their own regard, thus do present limitations to the amount of acceleration that can be realized. By focusing AX on projection and the underlying GEMM method calls, we will see that application utility lies within the span of possibility. Eq. 2 below is an expansion of Eq. 1 which clarifies the impact of related parameters on observable performance

$$\gamma^{-1} = \frac{Data_{up}}{\kappa * BW_{up}} + \frac{Data_{down}}{BW_{down}} + \frac{GFLOP_{func}}{GFLOP/s} + \frac{RTT}{\kappa} \quad (2)$$

where κ is the frame-face density (in faces/frame), *RTT* (round trip time) is the network *latency*, $Data_{up}$ and $Data_{down}$ are the uploaded data and downloaded results, along with the bandwidths of the uplink and downlink (BW_{up} and BW_{down}). $GFLOP_{func}$ and $GFLOP/s$ is the total number of GFLOPS required to execute the required functions and the computational speed (in GFLOPS per second), respectively.

To maximize γ we seek the maximization of each term's denominator (upload/download network transfer rates and computation rate). These are the most obvious. Notice that, by increasing the frame-face density κ, we can affect an increase in γ by decreasing the ratio of uploaded data to the complexity of summary computation. Maximization of γ may also be sought by minimizing the numerators in each term (upload/download data size as well as contributions from *RTT*). $GFLOP_{func}$ represents the computational *cost* of a subroutine, for our case, detection, projection, and search, which might be reduced by increasing algorithmic efficiency. Figure 2 shows achievable γ as a function of cloud compute capability, network data rates/latencies as well as frame-face density. Data sizes are as specified in Section II-B.

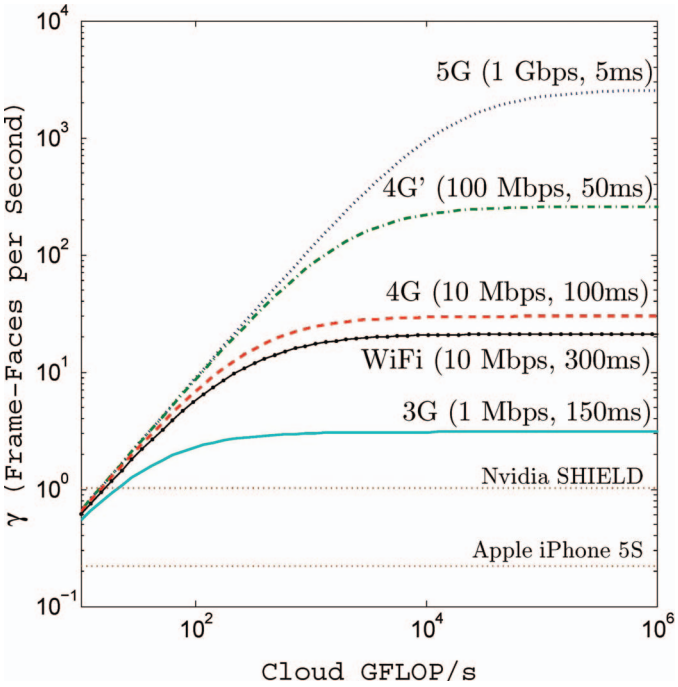

Fig. 2: Relationship between γ and cloud compute capability on various network generations, for FR application using a 5000-image database.

B. Limitations of AX

The most obvious impediments to real-time performance are the bandwidth limitations and latencies associated with the transport layer [37]. Considerations must be given to radio delays (i.e., 3G/4G/5G), IP Back-haul Transport latencies within the mobile service architecture and the connection between serving gateways (i.e., GGSN/PDN-GW) and AX domains. What Figure 2 illustrates clearly is that, 3G network speeds offer little to no acceleration potential for κ=1 (one face per frame) in comparison to local execution on a mobile device. Single-face FR requests processed over 3G on a cloud instance even with a PFLOP/s capability barely approach $\gamma = 0.5$, far from the baseline performance of the Nvidia SHIELD tablet. Each generation of network standards shows dramatic positive contributions to performance for any κ, given a reasonably powerful cloud (10–100 TFLOP/s). Furthermore,

each subsequent generation approaches a performance limit at much higher cloud compute capability, allowing for future exploitation of advancing computational resources.

C. Executing and Aggregating AX Requests

We consider a hypothetical cloud architecture of tightly coupled instances whose sole purpose is to execute AX Requests that are coming from *AX-aware* mobile application code. The nature of acceleration requires these computations to occur rapidly and relatively sparsely. We envision a large majority of these requests to execute on GPUs, as they can be utilized in a highly parallel fashion and offer competitive computational capacity for complex processes. Remote access to GPU clusters can be provided by routing traffic from TSP packet data network gateways (PDN-GW) over the internet to *AX servers* in the cloud. The Head Node in the cluster would be responsible for running the host-side AX requests and delegating requests to devices within the cluster architecture. Multiple AX requests could be aggregated and spread across available resources intelligently by the AX *execution run-time* to ensure minimal response times and fairness for each user. Pushing a request forward to a cloud instance with a device whose streams are already saturated would result in undesired delays. By adding machine instances to the AX pool, or increasing the efficiency of device utilization, the potential for reliable acceleration for any number of subscribers increases.

IV. AXaaS: Acceleration as a Service

Our AXaaS model formulates the computational acceleration (*AX*) for mobile applications as a billable service (*aaS*) by the TSP. AXaaS proposes to offer incentives to every participating party: I) TSP, II) consumer, and III) the developer, but begins with a sensitivity to the demands of **consumers**. Mobile device users simply want better applications and better performance. Developers need an architecture in which these applications and levels of performance can be realized. Cloud operators possess the computational resources to make this happen, however a unified structure does not exist among these entities with a focus on computation. Although this type of support can be implemented on an individual basis, we envision universal availability to enable the widespread adoption of the AXaaS model [38].

We conceptualize the TSP as the central component necessary for bringing AX to the mainstream. By renting cloud resources on behalf of customers, computational acceleration can be made available to millions of existing TSP customers through a simple augmentation of a monthly service agreement. As developers begin to understand how far the performance boundaries can be pushed, they will flood the market with applications and programming interface extensions and optimizations designed to take advantage of the AX potential. Spurring competition among developers to create and release the "next big thing" would ensure no shortage of options for the consumer. The increasing diversity and utility of applications will attract customers to AXaaS-enabled applications, thereby leading to service subscriptions.

An illustration of the relationship between the consumer, TSP, and cloud operator is given in Fig. 3. We see the TSP as a customer to the cloud operator, and as a service broker to its own customers (i.e., the consumer). This model is desirable since the TSP already possesses a vast customer base. It is relatively easy for the TSP to increase mobile device users' awareness of new products and services. Additionally, the TSP already possesses a robust communication network which reaches millions of mobile device users.

V. TSP view of AXaaS

The TSP could be considered as the arbiter of resource sharing and task allocation. By purchasing computing platform instances from cloud operators (e.g., AWS [18]), or using their own resources (e.g., Terremark [39]), a meta-device can be established whose sole purpose is to execute generic subroutines such as matrix multiplication in support of new compute-intensive applications. AX requests called through an API layer need only be routed through the existing wireless broadband network to an instance of AXaaS hardware.

A. Resource Acquisition

The Service Level Agreement (SLA) is centered on *availability* and *operational up-time* and must be extended to provide consistent and reliable *performance* in the acceleration of processes. Quality of Service (QoS) provisions seek to ensure the efficiency, speed and reliability of communication between entities over networks. Due to the nature of the proposed service model, QoS must also play a crucial role in ensuring maximum performance. As an example formation of an AXaaS cloud, we consider AWS configurable instance types [40], [41] and high compute-capable instances with GPUs such as a cluster of G2 (GPU Compute) instances which can be launched into a common placement group with low latency networking. For our cost analysis we will use instances launched in the U.S. East Region, and specify the instance types as Heavy Utilization Reserved [42].

B. Tiered Service Offerings

We conceptualize that TSPs offer AX in tiers using operation counts (FLOPs/mo), bundled with the existing data-throughput plans (GB/mo), each tier gradually increasing in cost and observable performance metric. We call upon the γ metric we introduced in Section II-C to assume this function, and frame our tier model around it. We suggest that each higher tier incorporates increasing QoS assurances for network performance (NQoS) and computation performance (CQoS), both of which limit the achievable values of γ. An example tiered plan is outlined in Table I with an outline of the evolution of advantages.

1) Tier I: Baseline tier for the casual user.
2) Tier II: Same as Tier I, except 2x volume and rate.
3) Tier III: Much higher computational volume and assured network rate, permitting much higher γ values.

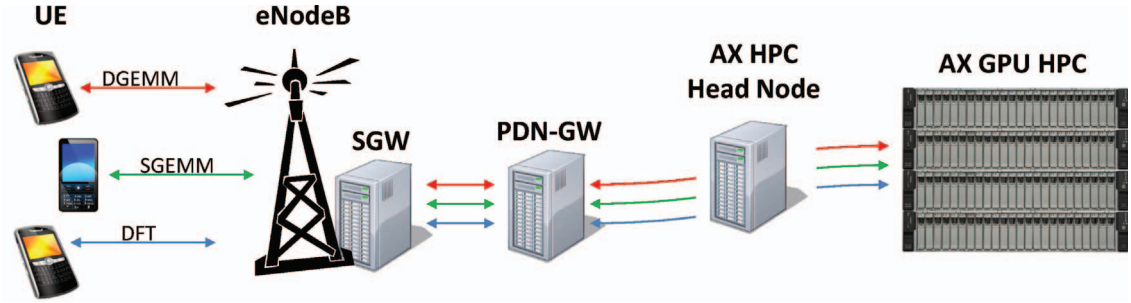

Fig. 3: The AXaaS model showing the basic AX transport scheme (AX Request and Replies). User Equipment (UE) communicates with LTE nodes (eNodeB) which tether to the IP backbone via Serving and Packet-Data Network Gateways (SGW/PDN-GW). IP traffic is then routed to the nearest AXaaS Head node which delegates resources in the GPU cluster to process AX requests. AX Replies are routed via the same path.

Tier	I	II	III	IV	V
Monthly Fee (F)	$10	$15	$30	$50	$100
TFLOPs Allocation (T)	10	20	40	80	160
GFLOP/s (CQoS)	80	200	2700	14K	27K
Mbps (NQoS)	5	10	25	50	100

TABLE I: Subscriber AX plan tiers showing subscriber cost, allocated operation count, Compute-Quality of Service guarantee (CQoS) and Network-Quality of Service guarantee (NQoS).

Distribution	Light	Medium	Balanced	Heavy
Monthly Revenue	$510k	$850k	$996K	$1.4M
Comp. Allocation	56.0%	98.7%	**134%**	**184%**
Gross Profit @ 27K	99.4%	99.6%	99.7%	99.8%
Marginal Sub. #	164	99	84	62
Users at 100%	48k	27K	20K	14.6K
Revenue at 100%	$909k	$850K	$737K	$734K
Profit at 100%	99.7%	99.6%	99.6%	99.6%

TABLE II: Revenue and Tier Distributions. *Light*: 90% of subscriptions in Tiers I, II, and III. *Balanced*: 20% subscription for each Tier. *Medium*: 50% of subscriptions in Tier III. *Heavy*: 80% of subscriptions in higher tiers.

4) Tier IV: Computational volume, rate, assured network rate, along with γ increase substantially.
5) Tier V: Max. computational volume and assured network rates for exceptionally demanding applications.

C. ROI Analysis

To provide an example of the effect of tier distribution we will take four distribution types: *Light, Medium, Heavy*, which are assumed to have concentrated subscriptions in the lower, middle, and upper tiers, respectively, and *Balanced*, which has equal subscriptions in all tiers. Refer to Table II which suppose there are 27,000 subscribers to a cluster with a maximum of 1.3 M allocatable double-precision TFLOPs per month (i.e., 1.3 dEFLOPs). Taking the *Medium* distribution as an example, we see that the TSP would need at least 99 subscribers to the cluster to break even (Marginal Sub. #). We also see that 27,000 subscribers in a *Medium* distribution across the tiers allocates 98.7% of the cluster compute capability in TFLOPs per month, and would generate approximately $850K in revenue at a gross margin of 99.64%.

In all cases we are seeing impressive potential revenues and favorable gross margins. It is clear that launching a greater number of clusters to service even more subscribers in multiple regions will result in increased architectural complexity, but also much greater revenues. This section is not meant to be an exhaustive ROI analysis, but rather an initial demonstration of the potential of the AXaaS model.

VI. End-User (Consumer) view of AXaaS

The impetus for a consumer to purchase a product involves its relative utility and cost. Since acceleration is tiered and priced by number of computations per month, the user must consider application use as an exhaustible resource. This way of thinking is quite familiar, as most data plans involve a finite capacity for data transfer. We suggest that this be a highly visible feature in each application (i.e., an AX-meter), integrated by developers, meant to enable a clear understanding of the application's consumption of *billable computation*. Aside from improving performance, AXaaS can also lower mobile power consumption as will be detailed below.

Mobile-Device-Only Energy Consumption: We will consider the iPhone 5S, whose battery capacity is approximately 6 Wh (1,570 mAh) [43], or 21,600 Joules. We estimated in Section II that the iPhone 5s processed a single FR routine in 4.6 seconds (i.e., $\gamma = 0.22$). The Apple A7 processor consumes 520 mA during floating-point operations (1.98 W @ 3.8 V battery voltage) [44]. This implies that the device would consume 9.09 Joules of energy in this 4.6 s face recognition interval. Therefore, the 21,600 Joule battery would be sufficient for 3 hours of continuous face recognition at the mere rate of $\gamma = 0.22$, which will not generate any significant amount of interest for this application.

Mobile+AX Energy Consumption: In this scenario, the mobile device would still consume energy to capture an image from its camera, send it to the cloud via the telecom link, and receive the results. We will pick the 4G scenario in Figure 2 (10 Mbps and 100 ms). To process the image in Figure 1 (left) ($\kappa = 13$), the phone would require 250 mJ of energy (116 mJ to send the 145 KB image over the 4G link, during which the power consumption is 1 W [45], and 50 mJ for image capture on highly-optimized hardware, and 84 mJ for idle time and to receive the results). This translates to a $\gamma = 39$ at a 0.75 W power consumption, and a battery life of 8 hours for cotinuous operation. These results show another favorable property of the AX-based acceleration: The *power efficiency* of the application changes from 9.09 Watts-per-γ to 0.019 Watts-per-γ , a 470x improvement (i.e., 1.98/0.22 vs. 0.75/39, respectively).

VII. Performance Evaluation

A. Experimental Setup

We profiled our FR application using NVIDIA Nsight Visual Studio Edition [46] on a x64 Windows 7 Pro workstation with an INTEL i7-4770K CPU @ 3.50GHz, 32 GB RAM and an NVIDIA GTX 760 GPU. The application was authored in C++ and compiled using Visual Studio 2010, OpenCV Library 2.4.8 and CUDA Toolkit v5.0. To supply the FR application with image data, we used an LG Nexus 4 running Android 4.2 Jelly Bean. The mobile device was mainly used for sourcing frames in our experiments, since the built-in face detection API only supports two faces per frame and is not sufficient for our tests.

B. Evaluation Criteria

The central focus of our experiments with regard to acceleration was the matrix multiplication (GEMM) component of the *Projection* routine, since it dominates the execution time. We had to consider memory bandwidth usage, task size in terms of number of computations and the kernel execution rate. The GTX 760 GPU we used in our experiments have nearly identical single- and double-precision floating point performance to the Nvidia GRID K2 units used in the AWS G2 cluster. Therefore, used the GTX 760-based results to estimate the AWS G2 instance performance. We extrapolated our results to the GTX Titan Z GPU when estimating the performance for tier III, IV, and V operation. We expect the TSP to utilize the lower-end GPUs only for acceleration types requiring heavy single-precision due to its lower cost, while GPUs like GTX Titan Z will be necessary for parts of the code that operate on heavy double-precision floating point data, such as the *Projection* phase of the FR application.

C. Experimental Results

Table III shows measured *Projection* performance for the GTX 760 along with intrapolations for the GRID K2, hence leading to our estimations for the AWS HPC [47]. Our estimations incorporate optimizations such as: 1) an omission of the eigenvector upload on each kernel launch, since a single upload can be amortized over a vast number of FR requests and 2) a full concurrency, where up to 16 independent streams may be executing simultaneously, which is a realistic assumption for a cloud environment utilizing heavy parallelism. Observe that a fully saturated GRID K2 can theoretically support a computational throughput of 200 FR Projections/s on a 5000-image DB. A cluster of 5xK2 devices augment this to roughly 1000 FR projections/s (last row) for the same DB.

Database Size	1000	2000	3000	4000	5000
760 PJ time (ms)	71.9	132.3	199.1	269.9	339.9
760 Stream #PJ/s	13.9	7.6	5.0	3.7	2.9
K2 PJ time (ms)	19.6	33.7	48.1	63.7	80.1
K2 Stream #PJ/s	51.0	29.7	20.8	15.7	12.5
K2 Device #PJ/s	815.8	475.1	332.6	251.4	199.6
5 x K2 Cluster #PJ/s	4079	2376	1663	1257	998.1

TABLE III: GK104 double-precision GEMM performance; *Projection* times and estimated executable projections/s.

Table IV is an extrapolation of the results in Table III and provides a list of AXaaS tiers with their network bandwidth and computational limits in the way previously described in Section V-B. Tiers I, II are assumed to utilize the Grid K2 GPU devices, and provide a network bandwidth guarantee (NQoS) of 5–10 Mbps, achieving γ=10–20, which is perfectly sufficient for the casual user. Tier II allows 2x the allocatable FLOPs, thereby allowing the user to enjoy 2x more runtime for the month of subscription. The "power tiers" (III–V) achieve much more impressive γ values in the 100–500 range and allow the user 16x more runtime for the month of subscription (i.e., 160 TFLOPs for tier V vs. 10 TFLOPs for tier I).

Tier	I	II	III	IV	V	Future
TFLOPs / mo	10	20	40	80	160	**400**
dGFLOP/s CQoS	80	200	2700	14K	27K	**100K**
Mbps NQoS	5	10	25	50	100	**1000**
Nwk Time (ms)	398	199	618	309	154	**18.6**
Exc Time (ms)	895	358	162	33	16	**4.3**
Req Time (ms)	1293	557	780	342	170	**22.9**
κ (faces/frame)	13	13	80	80	80	**80**
FR γ	**10.1**	**23.3**	**103**	**234**	**469**	**3493**

TABLE IV: FR performance for a 5000-image db, for Tiers I, II (κ=13), and Tiers III–V (κ=80), and predictions for a decade later.

Our final column looks forward, perhaps a decade into the future. Assuming that LTE-Advanced takes flight, offering full 1 Gbps transfer rates, and considering a cluster of 25 GPU servers, we estimate a potential $\gamma > 3000$. Within this time frame, an advanced mobile device such as iPhone is only expected to reach γ values of 1–2 from its current 0.22 based on its past decade's trend. Note that these results are meant to be no more than a back-of-the-envelope analysis to show the long-term business viability of AXaaS, rather than a complete business study. Final tiering of the product will entirely depend on the resources and business model of the TSP.

VIII. Conclusions

We proposed a new service architecture called Acceleration as a Service (AXaaS) whose central focus is on accelerating specific portions of resource-intensive mobile applications such as real-time face recognition (FR), real-time language translation and augmented reality. These applications spend a vast majority of their execution time on a very small set of API functions such as generalized matrix-matrix operations (GEMM) or Fast Fourier Tranform (FFT) operations. Therefore, to significantly improve application performance, only these API functions need to be executed much faster (i.e., *accelerated*) instead of the entire application.

We defined a metric γ that quantifies *user experience*, which is the *total number of faces recognized in a second* for the FR application. A similar metric to γ can be defined for almost every similar application. In the case of real-time FR, a standalone iPhone 5s can reach a γ=0.22, which is barely useful, if not *flat out boring*. When acceleration is applied, γ=10–20 values can be achieved, which will turn this application into a *useful* one, while γ=100 will make the application *wholly exciting*. On the other hand, γ=200–500

values will allow the application to be used in demanding business scenarios such as real-time surveillance.

We identified Telecom Service Providers (TSPs) as the most logical candidate for providing *acceleration* (abbreviated *AX*) as a service (aaS), hence our naming AXaaS. We showed that AXaaS could provide a new lucrative revenue stream and profits for TSPs when offered in different tiers, similar to today's tiered data services. We demonstrated that the AXaaS model is scalable, where an increasing subscriber pool does not degrade individual customer performances. Therefore, we conclude that AXaaS is a novel and viable business model today and will continue to be in the foreseeable future.

ACKNOWLEDGMENT

This work was supported in part by the National Science Foundation grant CNS-1239423 and a gift from Nvidia Corp.

REFERENCES

[1] E. Chen, S. Ogata, and K. Horikawa, "Offloading android applications to the cloud without customizing android," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, 2012, pp. 788–793.

[2] T. Soyata, H. Ba, W. Heinzelman, M. Kwon, and J. Shi, "Accelerating mobile cloud computing: A survey," in *Communication Infrastructures for Cloud Computing*, H. T. Mouftah and B. Kantarci, Eds., pp. 175–197. IGI Global, Sep 2013.

[3] Microsoft Corporation, "Mobile Assistance Using Infrastructure (MAUI)," http://research.microsoft.com/en-us/projects/maui/.

[4] C. Mei, J. Shimek, C. Wang, A. Chandra, and J. Weissman, "Dynamic outsourcing mobile computation to the cloud," *Technical Report TR11-006, University of Minnesota*, 2011.

[5] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*, 2011, pp. 301–314.

[6] E. Y. Chen and M. Itoh, "Virtual smartphone over ip," in *World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE International Symposium on a*, 2010, pp. 1–6.

[7] Y. Song, H. Wang, and T. Soyata, "Hardware and software aspects of vm-based mobile-cloud offloading," in *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, T. Soyata, Ed., pp. 247–271. IGI Global, 2015.

[8] T. Soyata, R. Muraleedharan, C. Funai, M. Kwon, and W. Heinzelman, "Cloud-Vision: Real-Time face recognition using a Mobile-Cloudlet-Cloud acceleration architecture," in *Proceedings of the 17th IEEE Symposium on Computers and Communications*, Jul 2012, pp. 59–66.

[9] B. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution.," .

[10] SYSTRAN Translation Tech., ," http://www.systransoft.com/.

[11] MS Translator, ," http://www.microsoft.com/en-us/translator/.

[12] Google Translate, ," https://translate.google.com/.

[13] M. Satyanarayanan, P. Bahl, R. Cáceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[14] T. Soyata, R. Muraleedharan, S. Ames, J. H. Langdon, C. Funai, M. Kwon, and W. Heinzelman, "COMBAT: mobile Cloud-based cOmpute/coMmunications infrastructure for BATtlefield applications," in *Proceedings of SPIE*, May 2012, vol. 8403, pp. 84030K–84030K.

[15] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Cloudlets: Bringing the cloud to the mobile user," in *Proceedings of the third ACM workshop on Mobile cloud computing and services*, 2012, pp. 29–36.

[16] N. Powers, A. Alling, R. Gyampoh-Vidogah, and T. Soyata, "Axaas: Case for acceleration as a service," in *Globecom Workshops (GC Wkshps)*, Dec 2014, pp. 117–121.

[17] MS Windows Azure, ," http://www.microsoft.com/windowazure.

[18] Amazon Web Services, ," http://aws.amazon.com.

[19] Google Cloud Platform, ," https://cloud.google.com/.

[20] NVIDIA CUDA Fast Fourier Transform (cuFFT) Library, ," https://developer.nvidia.com/cufft.

[21] NVIDIA CUDA Basic Linear Algebra Subroutines (cuBLAS) Library, ," https://developer.nvidia.com/cublas.

[22] AnandTech — The iPhone 5s Review, ," http://www.anandtech.com/print/7335/the-iphone-5s-review.

[23] NVidia Shield Tablet, ," http://shield.nvidia.com/gaming-tablet/.

[24] "General packet radio service (gprs) enhancements for evolved universal terrestrial radio access network (e-utran) access (release 8)," Tech. Rep., 3rd Generation Partnership Project, 2008, 3GPP TS 23.401 V8.1.0 (2008-03).

[25] Y. Song, H. Wang, and T. Soyata, "Theoretical foundation and gpu implementation of face recognition," in *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, T. Soyata, Ed., pp. 322–341. IGI Global, 2015.

[26] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, 2001, vol. 1, pp. I–511.

[27] M. Yang, D. Kriegman, and N. Ahuja, "Detecting faces in images : A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 1, pp. 34–58, 2002.

[28] Android API Reference, ," http://developer.android.com/reference/.

[29] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of cognitive neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[30] FaceRecognizer - OpenCV 3.0.0-dev documentation, ," http://docs.opencv.org/trunk/modules/contrib/doc/facerec/facerec_api.html.

[31] K. Kim, K. Jung, and H. Kim, "Face recognition using kernel principal component analysis," *IEEE Signal Processing Letters*, pp. 40–42, 2002.

[32] T. Soyata, *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, IGI Global, Aug 2015.

[33] O. Kocabas and T. Soyata, "Utilizing homomorphic encryption to implement secure and private medical cloud computing," in *IEEE 8th International Conference on Cloud Computing*, June 2015, pp. 540–547.

[34] O. Kocabas and T. Soyata, "Towards privacy-preserving medical cloud computing using homomorphic encryption," in *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, T. Soyata, Ed., pp. 213–246. IGI Global, 2015.

[35] A. Page, M. K. Aktas, T. Soyata, W. Zareba, and J. Couderc, "The QT Clock to Improve Detection of QT Prolongation in Long QT Syndrome Patients," *Heart Rhythm*, 2015.

[36] O. Kocabas, T. Soyata, J. Couderc, M. K. Aktas, J. Xia, and M. Huang, "Assessment of cloud-based health monitoring using homomorphic encryption," in *Proceedings of the 31st IEEE International Conference on Computer Design*, Oct 2013, pp. 443–446.

[37] M. Kwon, Z. Dou, W. Heinzelman, T. Soyata, H. Ba, and J. Shi, "Use of network latency profiling and redundancy for cloud server selection," in *Proceedings of the 7th IEEE International Conference on Cloud Computing*, Jun 2014, pp. 826–832.

[38] N. Powers and T. Soyata, "Selling FLOPs: Telecom Service Providers Can Rent a Cloudlet via Acceleration as a Service (AXaaS)," in *Enabling Real-Time Mobile Cloud Computing through Emerging Technologies*, T. Soyata, Ed., pp. 182–212. IGI Global, 2015.

[39] Verizon Terremark, ," http://www.terremark.com/.

[40] Amazon EC2 Pricing, ," http://aws.amazon.com/ec2/pricing/.

[41] AWS Amazon EC2 Instance Types, ," http://aws.amazon.com/ec2/instance-types/.

[42] Amazon EC2 Reserved Instances, ," http://aws.amazon.com/ec2/purchasing-options/reserved-instances/.

[43] The iPhone 5c and 5s have bigger batteries than the iPhone 5, ," http://www.gsmarena.com/the_iphone_5c_and_5s_have_bigger_batteries_than_the_iphone_5-news-6784.php.

[44] Google/Motorola Mobilitys Moto X Outpaces Competition with New Innovations, ," https://www.abiresearch.com/press/googlemotorola-mobilitys-moto-x-outpaces-competiti.

[45] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang, "Accurate online power estimation and automatic battery behavior based power model generation for smartphones," in *Intl. Conf. on Hardware-Software Codesign and System Synthesis (CODES+ISSS)*, Scottsdale, AZ, Oct 2010.

[46] NVIDIA Nsight Visual Studio Edition, ," https://developer.nvidia.com/nvidia-nsight-visual-studio-edition.

[47] High Performance Computing for Servers — Tesla GPUs — NVIDIA, ," http://www.nvidia.com/object/tesla-servers.html.