2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering

Avatar: Mobile Distributed Computing in the Cloud

Cristian Borcea, Xiaoning Ding, Narain Gehani, Reza Curtmola, Mohammad A Khan, Hillol Debnath
Department of Computer Science, New Jersey Institute of Technology
University Heights, Newark, New Jersey, 07102, USA
Email: {borcea, xiaoning.ding, gehani, crix, mak43, hd43}@njit.edu

***Abstract*—Avatar is a system that leverages cloud resources to support fast, scalable, reliable, and energy efficient distributed computing over mobile devices. An avatar is a per-user software entity in the cloud that runs apps on behalf of the user's mobile devices. The avatars are instantiated as virtual machines in the cloud that run the same operating system with the mobile devices. In this way, avatars provide resource isolation and execute unmodified app components, which simplifies technology adoption. Avatar apps execute over distributed and synchronized (mobile device, avatar) pairs to achieve a global goal. The three main challenges that must be overcome by the Avatar system are: creating a high-level programming model and a middleware that enable effective execution of distributed applications on a combination of mobile devices and avatars; re-designing the cloud architecture and protocols to support billions of mobile users and mobile apps with very different characteristics from the current cloud workloads; and explore new approaches that balance privacy guarantees with app efficiency/usability. We have built a basic Avatar prototype on Android devices and Android x86 virtual machines. An application that searches for a lost child by analyzing the photos taken by people at a crowded public event runs on top of this prototype.**

I. INTRODUCTION

Smart phones and tablets have become the personal devices of choice for most people, and huge amounts of data are generated daily by mobile users around the world. Users are generally willing to share and exploit this data within communities defined by friendship, similar interests, or geography for new and rich experiences. This scenario lends itself naturally to mobile distributed computing which enables direct collaboration among mobile users.

Existing solutions based on mobile ad hoc networks and Internet-based mobile peer-to-peer networks suffer from poor availability as mobile devices are not always on or reachable, limited and potentially costly network bandwidth, high latency, and reduced functionality due to limited computational, energy, and storage resources on the mobiles. Therefore, the question is: how to provide fast, scalable, reliable, and energy efficient distributed computing over mobile devices?

This paper proposes Avatar, a novel system which can achieve these goals with help from the cloud. Avatar is a mobile-cloud system that enables effective and efficient collaborative apps for mobile users and provides the following features: (1) a high level programming model that is simple and flexible; (2) high availability for user apps/devices; (3) low latency/response time for apps; (4) high scalability for the cloud infrastructure; (5) isolation and effective resource management for mobile user apps in the cloud; (6) resource savings on the mobiles; and (7) mobile data privacy.

In Avatar, a mobile user owns one or more mobile devices and an associated "avatar" hosted in the cloud, as shown in Figure 1. An avatar is a per-user software entity which acts as a surrogate for the user's mobile devices to the extent possible, thus reducing the workload and the demand for storage and bandwidth needed on the mobiles. The avatars are instantiated as virtual machines (VMs) in the cloud in order to provide resource isolation and to simplify per-user resource management. Avatars run the same operating system as the mobiles and can thus run unmodified app components. Implicitly, they save energy on the mobiles and improve the response time for many apps by executing certain tasks on behalf of the mobiles. The avatars are always available, even when their mobile devices are offline because of poor network connectivity or simply turned off. Four of the features listed above (high availability, low latency, resource isolation, and resource savings on mobiles) are implicitly offered through the Avatar concept. The other three features, i.e., programming model, cloud scalability, and privacy, represent the major research challenges of this system.

Programmability Challenge: Unlike previous research that has focused on programming stand-alone mobile-cloud apps [1]–[3], our research investigates distributed mobile-cloud apps. Avatar apps execute over distributed and synchronized *(mobile device, avatar)* pairs that cooperate to achieve a global goal. App components have multiple options to divide the execution among mobile devices and avatars to achieve different global performance objectives. However, the programming abstractions should shield the programmers from this complexity and provide a simple high level API. In addition to the app code, the programmer needs only to provide policy and performance objectives which will be translated into an execution plan by the Avatar middleware.

Cloud Scalability Challenge: So far, the research community has not investigated the impact on the cloud architecture and protocols of supporting billions of mobile users, which includes executing mobile app components in the cloud. The mobile distributed apps will generate traffic and system load which are significantly different from the current cloud workloads. For example, the data generated by mobile devices is usually redundant, unstructured, and dispersed around the cloud. These apps are highly diverse, and most of them do not run continuously. Many apps are interactive or heavy on communication instead of computation. Therefore, new cloud architectures and protocols are needed to maximize scalability and find a good balance between cost and efficiency.

Privacy Challenge: Finally, we acknowledge that Avatar could lead to "big brother" scenarios: since the applications or parts of them will execute in the cloud, they would need access to unencrypted user data. The challenge then is how to use the cloud to store data and efficiently execute apps while guaranteeing that the cloud provider cannot access the protected data and the results of processing this data. Thus, we need to examine if existing approaches to protect privacy in the cloud are appropriate and, if not, to explore new approaches that balance privacy guarantees with app efficiency/usability.

II. AVATAR ARCHITECTURE OVERVIEW

In our architecture (Figure 1), there is one avatar in the cloud for each user. All mobile devices of a user are associated

978-1-4799-8977-5/15 $31.00 © 2015 IEEE
DOI 10.1109/MobileCloud.2015.22

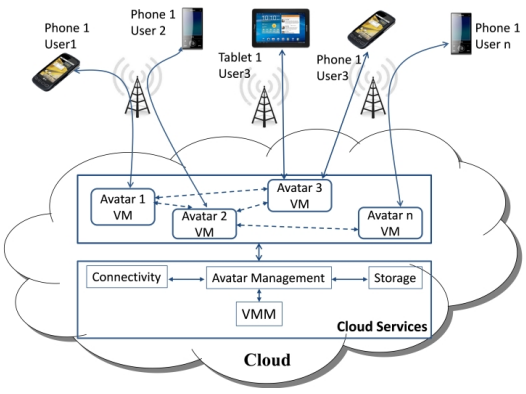

Fig. 1: Avatars in the cloud enable mobile devices to run novel collaborative mobile distributed apps

with the same avatar. The avatar is a virtual machine (VM) that runs user apps or app components. This VM also runs a daemon program for synchronization with the mobile devices of the user. A number of cloud services provide support to avatars (communication, storage, offloading, and avatar management) to enable effective collaboration with high efficiency. While logically the avatars could be hosted by any cloud provider, we expect mobile network operators to offer this service since they have already started to offer cloud services similar to iCloud to back up data from mobile devices. Having information about the mobile devices and their users, the mobile network operators may offer better services with higher efficiency.

Each avatar coordinates with its mobile devices to synchronize data and schedule the computation of avatar apps or app components on the avatar and/or mobile devices. In addition to maintaining synchronization with the mobiles, the daemon program in the avatar hides the complexity of the underlying system (e.g., name resolution, low-level communication details) to provide high-level system support to apps and simplify programming. With this support, the app instances running in avatars and mobile devices can collaborate with each other to perform distributed computing. A mobile device does not interact directly with the avatars of other mobile users. User to user communication always goes through the avatars.

III. Related Work

Cloudlets [4] and several types of applications that leverage them [5] represent an orthogonal research area to Avatar. Our system can take advantage of the cloudlet ideas to further optimize the performance (e.g., latency).

Offloading mobile code execution to the cloud has been extensively studied recently [1]–[3], [6]. While we leverage results from this work, our goal is fundamentally different: we aim to provide novel collaborative distributed apps executed efficiently (i.e., with help from the cloud) in a peer-to-peer fashion among mobile users. This goal brings new programming challenges and optimization opportunities. In addition, Avatar maintains an always-on per-user entity in the cloud (i.e., the avatar), which improves app availability, allows for user-to-user collaborative apps, and provides support for per-user resource management in the cloud.

Very recently, a few works have investigated cloud support for mobile distributed computing. Clone2Clone [7] offloads peer-to-peer networking to the cloud. The research in [8] proposes a cloud entity that acts as a communication endpoint for all mobile devices of a user. VehiCloud [9] offers a similar idea to enable routing in vehicular networks. These projects focus mostly on enabling communication among mobile users. Avatar supports this functionality as well, but in addition, it provides full system support for execution of mobile distributed apps, which addresses distributed programming, availability, scalability, and privacy features.

The work in [10] developed a component based approach where application developers define software components with their dependencies, configuration parameters, and constraints. These components are then distributed and configured at runtime for optimization. Avatar is developing similar optimizations for an environment which is by default distributed, as an application leverages the mobile devices and avatars of many users. However, due to cloud scalability and privacy constraints, the actual solutions will be different.

Sapphire [11] is a distributed programming platform that separates the application logic from the deployment logic. Thus, programmers can modify distributed application deployments without changing the application code (e.g., change the caching behavior or the fault tolerance mechanism). This work could be leveraged in Avatar to allow for dynamic management of non-functional application features.

IV. Applications

Avatar enables many novel mobile distributed apps and makes existing apps more efficient.

Finding people of interest in a crowd: A parent could search for a lost child using the child's photo to search among recent photos taken by nearby mobile users in a crowded tourist area. The *find person* app could run in parallel on either the users' avatars or their mobiles depending upon where the photos are located and the latency trade-offs between computation and communication. Fast response time is of essence here, and our system improves the response time by using avatars to process the photos already uploaded and by deciding how to deal best with the photos residing on mobiles.

Healthcare and wellbeing: Users may have health-monitoring body sensors which report data to smart phones and then on to the avatars; additionally, the phones may record the user location and their co-location with other users. A simple example app is one that would allow users or health agencies to monitor and stop, in early stages, the spread of epidemic diseases. The user mobiles and avatars may collaborate regionally to make localized decisions for improved scalability and privacy. In a disaster situation such as an earthquake, the mobiles/avatars of users could be queried in real-time from mobile devices carried by emergency teams to find the places where vital signs have been located. The avatars may share the users' data even after the mobiles have run out of battery power, thus improving availability. In all these healthcare apps, the users control who can access their privacy-sensitive data.

Collaborative mobile sensing and filtering: As mobile people-centric sensing is becoming more widespread, machine learning algorithms are being used to classify the sensed data. The mobiles and the avatars can run collaborative voice/speech recognition. Our architecture allows for faster response times (due to the higher computation power of the avatars) and energy savings on mobiles.

Mobile multi-player gaming: As an example, consider a fast pace game such as Quake 3 (first person shooter

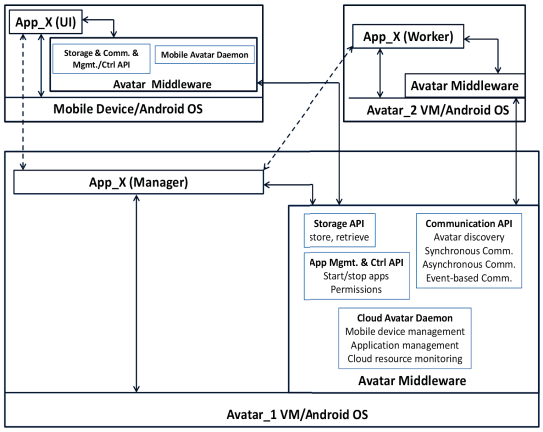

Fig. 2: Avatar middleware and example of app execution

game) which was prototyped in P2P settings [12]. The updates are sent between avatars, which then compute the scenes. Many optimizations that require significant computation and communication can be done by avatars, thus improving the overall mobile experience. Avatar has many benefits in this example: improved user experience (i.e., no need to do expensive computations on the mobiles), large energy savings on mobiles, significant reduction of wireless bandwidth consumption.

V. Programming Model

Our programming model must deal with two unique features of mobile distributed computing with avatar support. First, the computing is unstructured and decentralized. Though avatars are always on, they are associated with dynamic mobile users and enable dynamic user collaboration. *The users become computing entities* in the sense that they produce data, provide context information, and interact with distributed apps. This is different from traditional distributed computing where machine-to-machine cooperation is well structured and managed by system administrators or programmers. Second, data and computation must be carefully managed to deal with heterogeneous computing devices and to meet privacy requirements. Each app component can run at several places — avatar and one or more mobile devices of the user. This is also different from traditional distributed computing where an app component is bound to one node. In addition, the app cannot save private data or schedule computation on computing devices of other users for security reasons. Differently, traditional distributed computing can schedule computations on any node that has the required resources.

A. Avatar App Structure

An avatar app can be constructed in a 'SPMD' style — all app instances run the same code, but the computation and the data being processed may differ on different devices. If we assume that each user has one mobile device, the application is partitioned (statically or dynamically) in two parts: one that runs on the mobile device, and one that runs in the avatar. While the application is the same for each user participating in the distributed computation, the way it is partitioned differs for each user. As shown in Figure 2, the avatar is a container for apps or app components running in the cloud (i.e., a VM) and, at the same time, runs a daemon program to synchronize with the mobiles (with a daemon instance in the cloud and one each on the mobiles). Avatar VMs will run the same OS with mobile devices such that apps are able to run unchanged on both avatars and mobile devices. A common avatar API is exposed to the apps on both mobile devices and avatars by an Avatar middleware. Note that non-Avatar apps (i.e., current mobile apps) are not impacted by Avatar and can run as usual.

To better understand Figure 2, let us consider a simple "Lost Child" app: a parent could use it to locate a child lost in a crowded area using the child's photo to search among recent photos taken by nearby mobile users. The programmer organizes this app in 3 components: (1) UI which runs on the mobile; (2) Manager which runs in the avatar of the lost child's parent; (3) Workers which run in the avatars of people who will help to recognize the image of the child in the photos recently taken by the users associated with these avatars.

The app starts executing with the UI component on the mobile of the parent searching for the child. This component receives as input the photo(s) of the child as well as the time/location where the child was lost. Then, it interacts with the Manager component on its associated avatar. The Manager uses the API to invoke the local avatar to find the avatars that satisfy the spatio-temporal constraints and which have this app. The avatars that are willing to collaborate, start the Worker component of the app which performs face extraction from its available photos and then face recognition. Thus, the distributed application runs in parallel on these avatars (and potentially on their associated mobile devices) as well as the mobiles/avatar of the user who initiated the application. Each Worker component returns the location and time where/when the photos most similar with the lost child have been taken. The actual photos could be returned as well. The Manager component aggregates the answers and sends them back to its mobile (i.e., the mobile of the parent who activated the app).

The "Lost Child" example illustrated a one-to-many communication pattern, but Avatar can be used in the same way for a many-to-many communication pattern. In such a case, avatars will run both the Manager and the Worker components of the app. There are situations when the Worker component has to run on the mobile devices (e.g., apps that must read data from the phone sensors). This functionality is handled by the avatar daemon at the mobile. Let us note that communication and storage are handled transparently by the Avatar middleware independent of where the app component runs.

The apps can be statically or dynamically partitioned. The above example sketches a basic programming model with static partitioning, which is done by the programmer. Dynamic partitioning can be done based on certain pre-set rules. For example, apps can be partitioned automatically between the avatar and the mobile devices based on user preferences (e.g., privacy), resource limits, and performance parameters in order to achieve a good trade-off between various goals. Unlike existing partitioning schemes for stand-alone apps, the partitioning in Avatar is more complex because it considers the graph of the distributed entities (i.e., mobile devices and avatars involved in the distributed computation), rather than just two entities (i.e., mobile app on the phone and its counterpart in the cloud).

B. API and Middleware

The Avatar programming model provides the same API for the app instances on avatar and mobile devices. This effectively hides heterogeneity and programming complexity, and it also

effectively supports dynamic app partitioning. For everything other than avatar operations (including communication with various services over the Internet), the apps will use the local platform (e.g., Android) API.

The middleware includes the avatar daemon and the mobile daemon. These daemons coordinate with each other to maintain synchronization between the mobiles and their avatar. For example, the mobile daemon monitors local resources (e.g., battery) and location, and it updates the avatar daemon as needed. They also coordinate to decide the currently active mobile device. The avatar daemon enforces common access control policies for all user apps and monitors the user resource consumption in the cloud.

Communication API: The Avatar platform provides three types of communication: synchronous (send/receive), asynchronous using a key-value data store, and event-based (publish/subscribe). For synchronous communication, the programmers will use abstract avatar IDs for communication. These IDs can be obtained in a straightforward way from the cloud provider in certain situations. For example, in the "Lost Child" app mentioned above, the cloud provider which is also the wireless carrier can easily find out which users are in the region of interest during the specified time, and then return the IDs of their avatars. Other apps, however, locate avatars based on additional properties such as social connections or type of capabilities/data they possess (e.g., sensing). In this case, we employ P2P techniques to provide scalable name resolution.

Since many cloud providers offer key-value data stores, we leverage them to provide asynchronous communication. Synchronization in distributed apps is achieved through blocking *get* calls in the key-value data store; the app unblocks when a *put* call on the key succeeds. Existing cloud notification services are leveraged to implement event-based communication.

Storage API: It allows apps to persistently store data in the cloud, on the mobiles, or both. In the cloud, data is stored in persistent object storage such as Amazon S3 or in block storage such as Amazon EBS. The daemons, at the mobiles and at the avatar, monitor the data stored in all places and synchronize it.

App Management API: It allows apps to start/stop app components and to verify access control permissions.

VI. Cloud Architecture

The avatar cloud infrastructure provides and manages computing resources, storage space, and interconnection required by avatars. Conventional general-purpose cloud infrastructures, e.g., those hosting virtual desktops or supporting VM clusters for data processing, cannot satisfy the demand of Avatar workloads on system scalability and efficiency. The Avatar cloud infrastructure must be designed to fully consider and leverage the features of avatars and mobile distributed computing.

Firstly, scalability and resource efficiency are the most important design considerations. Due to privacy reasons, an avatar cannot be shared by multiple users. Given the huge number of mobile users, the Avatar cloud will be required to deal with billions of VMs, one for each avatar. Secondly, the Avatar cloud will have to support mobile devices and mobile distributed computing. For an avatar, its workloads and its communication patterns with other avatars vary over time, determined by the demands of its associated user. Compared to conventional clouds, where each VM usually has specialized tasks (e.g., providing a web service or running Hadoop tasks), the avatar cloud must have higher ability to tolerate and efficiently support such workload dynamics in avatars. Finally, each avatar belongs to a different mobile user. It is not possible to enforce centralized management and coordination. Thus, their data and the computation on the data can be highly redundant. This can significantly increase system resource consumption (e.g., storage space and CPU usage). Nevertheless, it provides opportunities to improve efficiency and performance by removing the redundancy.

To develop a customized cloud infrastructure for avatars, we explore the following techniques and design choices.

Virtual machine clustering to localize communication: Data centers usually employ hierarchical networks. Poor scalability is caused by inefficient utilization of the backbone network at the top layer [13]. VM layout impacts system scalability by affecting the amount of data traffic on the backbone network. Virtual machine clustering improves VM layout by putting collaborating VMs close to each other on the network to localize communication for better system scalability.

However, due to the workload dynamics, the interaction between VMs changes over time. This requires the clustering to be adjusted dynamically by moving VMs. It not only increases the difficulty of VM clustering, but also the extra overhead incurred by moving VMs can offset the benefit of clustering. To address the problem, we first cluster avatars (i.e., VMs) based on their user's geographical locations, social connections, and contact lists when they join the system. This basically sets a "home" for each VM where it often interacts with its neighbors, and reduces the chance that a VM moves to far away locations. Then, we employ a few techniques to further reduce the movement of VMs and their data when adjusting VM clusters, which are introduced below.

Distributed storage and data layout to localize data accesses: Localizing data access reduces remote data access and the associated network traffic. This not only helps improving system scalability, but also improves the performance of each individual app by reducing I/O latency. Using distributed storage and matching data layout with VM layout are the keys to localize data accesses.

We use locally shared storage systems (SAN or NAS) for each swamp of servers (e.g., servers on the same rack or on the same subnet). We choose this architecture because of its advantages in reducing VM migration costs and reducing storage space consumption: 1) local VM migrations do not need to move VM images; 2) migrating VM images can be carried out lazily for remote VM migrations; and 3) data redundancy can be effectively removed in each local storage. Conventional distributed storage design with local disks attached to every computing server cannot offer these benefits.

To reduce the cost of data movement and/or remote data accesses due to VM migrations to remote storage systems, we separate the data sets in each VM image based on which apps access them and duplicate/distribute the data sets to the storage systems where the VM usually runs the apps.

Reducing data and computation redundancy: In avatars, there will be much data redundancy (e.g. shared pictures and video clips). For a system serving billions of users, data redundancy must be removed to reduce storage space consumption. Deduplication has become a mature technique to effectively remove data redundancy in secondary storage systems. However, when it is applied to primary storage, maintaining high I/O throughput becomes a challenging issue,

because deduplication may delinearize data placement and remove local data copies. The large number of avatar VMs requires high I/O throughput, in addition to the huge storage space. To ease the tension between I/O throughput and storage space, we avoid global deduplication across storage systems and avoid deduplicating data sets that are frequently used.

Redundant data may lead to redundant computation, increasing the burden on the servers. To reduce redundant computation, we cache and reuse computation results on redundant data. For example, a user may send a picture of a person and ask her friends if any one of them knows the person. The avatars of her friends will compare the face in the picture against the faces stored by their avatars. In this case, there is no need to repeat the comparison on shared pictures.

Schedule VMs and requests carefully to further reduce computing resource consumption: With a huge number of avatars in the cloud, each cloud server must be densely packed. Thus, it is important to efficiently utilize hardware resources on each individual server. Because avatars will become idle when they do not have requests/tasks to process, the general guidelines to save hardware resources are 1) put idle avatars to sleep to reduce resource consumption, and 2) without degrading performance (e.g., response time), maximize the time that avatars spend on sleeping.

To minimize the interruption to sleeping VMs, we batch and re-route requests to these VMs. *Request batching* marks requests with soft real-time deadlines. As long as their deadlines have not arrived, requests are queued and sent to the avatar in a batch when resumed. *Request re-routing* is based on the observation that anycast requests can be directed to any avatar with the same content/context-based name (e.g., avatars running the same app in the same geographical location). Thus, such requests can be re-routed to light-loaded active avatars with the same data content to reduce the delay and overhead caused by resuming a sleeping avatar VM.

VII. Privacy

In the proposed Avatar architecture, all user data is stored at a single Cloud Service Provider (CSP), which is typically the user's mobile network operator. Some users may not be willing to participate in Avatar because they do not want to give the mobile carrier unfettered access to their sensitive data. While there are a few possible solutions, they have various limitations as summarized below. Therefore, we propose a different solution that extends the Avatar architecture with protocols that offer an enhanced level of data privacy at the cost of running an additional avatar per user.

An alternative solution would be the use of homomorphic encryption, which allows for computations over encrypted data. Despite significant progress [14], existing solutions based on this method are either inefficient or inflexible (e.g., provide programming support for a limited set of operators).

Another potential solution is to use hardware support to ensure shielded application execution over untrusted cloud platforms. For example, Haven [15] extends the hardware level protection features provided by the Intel SGX architecture from code snippets to the entire OS. But there are limitations: this solution slows down the computation substantially; the entire security architecture depends on the chip manufacturer's ability to protect the secret keys; programmers will miss certain features, such as process creation, that are not supported; the architecture does not support multiple processes per VM.

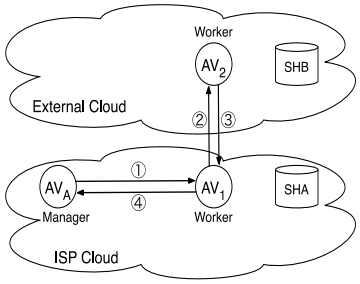

Fig. 3: The privacy architecture of Avatar

Can we use existing secure multi-party computation techniques? A user's avatar engages in a protocol with avatars of other users in order to perform a function that receives as inputs the user's data and the data of the other users. A natural approach would be to apply one of the many results from the area of secure two-party computation in an attempt to preserve the privacy of users' data: A client and a server, each holding a private input, engage in a privacy-preserving protocol at the end of which they do not learn any information except the outcome of the function on their private inputs. Some solutions are general and can compute any function in a privacy-preserving manner [16]; alas, they are quite inefficient. More efficient protocols have been designed for privacy-preserving set operations, such as set intersection [17] or set disjointness [18].

Such existing privacy-preserving protocols are not suitable because the setting we are considering is different. The server is conceptually represented by the collection of avatars belonging to users who participate in the system and make their data available. Each avatar runs on an untrusted third party CSP. As such, a direct application of previous approaches will mean that the CSP who runs the server will have access to the data of all users who participate in the system. From a privacy perspective this solution is not satisfactory.

A new approach. Instead, we propose a new model: (1) store the users' data split between two different CSPs in such a way that each individual CSP can access only a split version (semantically meaningless) of the user's data stored in its cloud; (2) have one avatar in each of these CSPs for each user; (3) compute the desired function as a protocol executed between the user's avatars. The privacy of the users' data is preserved as long as the CSPs do not collude with each other.

Figure 3 illustrates our model. We assume that each user U who participates in the system has access to the services offered by two CSPs: their own mobile carrier cloud (CSP_1) and one other external CSP (CSP_2). Each user U runs two avatars instead of one, AV_1 on CSP_1, which acts as primary avatar where all the requests are directed by default, and AV_2 on CSP_2. User U splits her data in two shares SHA and SHB and stores SHA at CSP_1 and SHB at CSP_2. The splitting is done according to a *splitting function* defined by the app programmer. Assume user A (*i.e.*, the requester or manager) and user U (*i.e.*, the worker) want to compute a function which uses input from both A and U. The following steps are executed in order to preserve the privacy of U's input.

In Step 1, A's avatar AV_A, acting as a manager, sends A's input to AV_1 (which is U's avatar located at the same CSP as AV_A). In Steps 2 and 3, AV_1 engages in a protocol with AV_2, in order to determine the outcome of the desired function on A's and U's data. AV_1 and AV_2 act as workers executing a

task given by the manager AV_A. To describe the exact details of the computation and the protocol between AV_1 and AV_2, the app programmer must define a *compute function*. Just like the splitting function, the compute function will be specific to each app. Finally, in Step 4, AV_1 returns the result of the desired function to AV_A.

We believe this approach of splitting the data between two (or more) clouds can result in privacy-preserving protocols that are more efficient than those based on the traditional secure multi-party computation described above. The assumption that the clouds do not collude with each other in order to break the privacy of their users is supported by a real-world setting in which the CSPs are competitors (e.g., Amazon and Microsoft).

VIII. Current Prototype

We have built a basic prototype of the Avatar platform and an avatar app. We use Android based mobile devices (both mobile phones and tablets) all of which can connect to the Internet via WiFi and/or cellular networks. Each avatar has an Android-x86 VM [19] with a single virtual processor and 4GB memory (similar to m3.medium Amazon instances). We build the images in a way that they can run on both the Amazon AWS and on our local servers.

We have implemented the "Lost Child" app discussed earlier. To identify the avatars which may have photos of the child, a directory service/name resolution is needed to find out who was close to the place where the child was lost during the time when this happened. In the current stage, we rely on a dummy server to provide the IDs of the avatars of interest. When these avatars receive the request, they use image recognition algorithms to examine the pictures that possibly have the lost child's face. If positive results are observed in some pictures, the avatars send back the location and shooting time of these pictures. Then, the avatar of the user who initiated the app forms a trajectory of the child movement, which is shown on a Google map on the mobile. To minimize false positives and false negatives in face recognition, we run three face recognition algorithms (Eigenface, Fisherface and LBPH) and use majority voting to decide the best results. These algorithms are implemented in the OpenCV library and exported to the app using Java wrappers. Apart from face recognition, we also run background tasks for face detection, which crop faces from the pictures and prepare the face databases.

We implemented a simple partitioning mechanism for image recognition tasks, which is directed by user defined policies. For this app, the user specifies that the tasks should run on the entities where the pictures can be accessed. If the pictures are available on the avatars, then the task runs there; otherwise, for the images that are kept on the phones for privacy reasons, the task runs on the phone.

IX. Conclusion

This paper has introduced the Avatar system for mobile distributed computing enabled by a new cloud architecture. The goal of this system is to solve a long-standing problem: how to achieve the true potential of mobile distributed computing despite its many challenges such as resource limitations, mobility, and sensitive nature of the mobile data? Our solution relies on associating all mobile devices of a user with an avatar in the cloud, and dynamically deciding the app execution plan on top of a distributed environment consisting of avatars and mobile devices. In addition to the programming and middleware challenges, we are also addressing cloud architecture and privacy issues to make the Avatar system compelling to users, ISPs, cloud providers, and app developers.

Acknowledgment

This research was supported by the National Science Foundation (NSF) under Grants No. CNS 1409523, CNS 1054754, and DUE 1241976. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," *Proceedings of the 6th EuroSys Conference (EuroSys 2011)*, pp. 301–314, 2011.

[2] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," *Proceedings of the IEEE Infocom 2012*, pp. 945–953, 2012.

[3] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," *Proceedings of the 8th international conference on Mobile systems, applications, and services (MobiSys '10)*, pp. 49–62, 2010.

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

[5] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan, "Scalable crowd-sourcing of video from mobile devices," in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '13. New York, NY, USA: ACM, 2013, pp. 139–152.

[6] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: ACM, 2011, pp. 43–56.

[7] S. Kosta, V. C. Perta, J. Stefa, P. Hui, and A. Mei, "Clone2clone (c2c): Peer-to-peer networking of smartphones on the cloud," in *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing*. Berkeley, CA: USENIX, 2013.

[8] K. Kim, S. Lee, and P. Congdon, "On cloud-centric network architecture for multi-dimensional mobility," *Computer Communication Review*, vol. 42, no. 4, pp. 509–514, 2012.

[9] Y. Qin, D. Huang, and X. Zhang, "Vehicloud: Cloud computing facilitating routing in vehicular networks," *Proceedings of the 11th IEEE TrustCom*, pp. 1438–1445, 2012.

[10] T. Verbelen, P. Simoens, F. De Turck, and B. Dhoedt, "Leveraging cloudlets for immersive collaborative applications," *Pervasive Computing, IEEE*, vol. 12, no. 4, pp. 30–38, Oct 2013.

[11] I. Zhang, A. Szekeres, D. V. Aken, I. Ackerman, S. D. Gribble, A. Krishnamurthy, and H. M. Levy, "Customizable and extensible deployment for mobile/cloud applications," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. Broomfield, CO: USENIX Association, Oct. 2014, pp. 97–112.

[12] A. Bharambe, J. Douceur, J. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: enabling large-scale, high-speed, peer-to-peer games," *Proceedings of the ACM SIGCOMM 2008 conference on Data Communication*, pp. 389–400, 2008.

[13] "Amazon ec2 performance drops too many users." [Online]. Available: http://www.thebuzzmedia.com/amazon-ec2-performance-drops-too-many-users/

[14] "MIT Technology Review, Homomorphic Encryption: Making Cloud Computing More Secure." [Online]. Available: http://www2.technologyreview.com/article/423683/homomorphic-encryption/

[15] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," in *Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation*. USENIX Association, 2014, pp. 267–283.

[16] A. Yao, "How to generate and exchange secrets," *Proc of FOCS*, pp. 162–167, 1986.

[17] M. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," *In Advances in Cryptology EUROCRYPT*, 2004.

[18] A. Kiayias and A. Mitrofanova, "Testing disjointness of private datasets," *Proc of Financial Cryptography (2005)*, p. 109124, 2005.

[19] "Android x86." [Online]. Available: http://www.android-x86.org/