The Case for a Local Tier in the Internet of Things

John Kolb
Kaifei Chen
Randy H. Katz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-222
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-222.html

December 30, 2016

The Case for a Local Tier in the Internet of Things

John Kolb, Kaifei Chen, Randy H. Katz
Computer Science Division
University of California, Berkeley
{jkolb, kaifei, randy}@cs.berkeley.edu

ABSTRACT

The Internet of Things, characterized by hardware heterogeneity, siloed hardware and software platforms that stifle interoperation, and systems that are distributed by nature, represents a challenging domain for application developers. Many industrial and academic efforts have produced tools seeking to remedy this situation, but they typically operate at a low level of abstraction centered on individual devices or give developers little control over the deployment and execution of their applications. Most IoT development frameworks are designed around the notion of integrating connected edge devices with cloud-based services, ignoring an intermediate collection of local resources, such as workstations and servers, that can help satisfy many applications' requirements while also protecting user privacy by keeping sensitive data off of the cloud. We propose a distributed IoT runtime system to assist developers in the creation, deployment, and management of their applications. This system is intended to incorporate edge devices, local resources, and cloud servers while also giving developers precise control over where and how their software is executed. We motivate our system through a case study accompanied by microbenchmarks, present a preliminary system design, and discuss the issues raised by our proposal.

1. INTRODUCTION

Developing applications for the Internet of Things (IoT) involves several challenges. These applications must deal with highly heterogeneous hardware, which can range from low power embedded microcontrollers deployed at the edge of the network to powerful servers running in the cloud. The Internet of Things also suffers from a lack of interoperability among both hardware and software systems. Many devices are only accessible through vendor-specific APIs or software platforms. Finally, because IoT applications are running on distributed systems, the developer must pay particular attention to where computation will be performed and where data will be stored. The situation is further complicated by the fact that IoT devices and users are frequently mobile, making location a primary concern in IoT apps. Decisions on where to place data and computation have important consequences for performance, security, user privacy, and reliability. For example, the cloud offers practically limitless storage and computation, but also suffers from potentially high latency and inconsistent network bandwidth. Use of the cloud also raises privacy concerns, as users may be opposed to the use of the cloud to store their sensitive data.

Numerous IoT development toolkits and frameworks have been proposed in recent years. However, we believe they inadequately address the problems cited above. While a plethora of platforms have been put forth by industry, the majority are mainly concerned with agreeing upon common interfaces and protocols to ensure mutual compatibility between different products. These platforms force developers to reason at the low level of individual devices and do little to ease the task of composing these devices into a meaningful application or service. The platforms that do provide tools to help developers, such as a mechanism for device discovery, also lock these developers into a particular set of vendor-specific technologies. Perhaps more importantly, these frameworks tend to reduce the Internet of Things to a two-layer entity: a layer of relatively weak edge devices to collect data and perform actuation coupled with a layer of more powerful servers running in the cloud. This overlooks the critical role that a third, intermediate tier of local resources, such as personal workstations and building servers, can play in the Internet of Things. We have dubbed this the “stratus” tier, after the low altitude clouds.

We are also beginning to see academic efforts in the IoT space that are more explicitly concerned with application development, e.g. [16], but they either omit the stratus tier or fail to give the developer control over how their applications run on the underlying infrastructure. Ultimately, today's IoT frameworks represent two extreme situations, neither of which is desirable: they either force developers to write applications primarily focused on hardware devices and their low-level interactions, or they offer a higher level of abstraction but wrest control over where computation occurs and where data is stored away from the developer and thus the end user.

We believe that several fundamental improvements to the state of the art in IoT application development are necessary in order for the Internet of Things to achieve its full potential. Developers need a set of tools that will enable them to rapidly create non-trivial applications incorporating multiple hardware and software systems. Additionally, these tools must give the developer full control over where and how their applications run on the underlying infrastructure. This facilitates the creation of applications that in turn allow end users to control the propagation of sensitive data.

In this paper, we present our vision for a distributed IoT runtime system that addresses these issues. Our proposed system is centered on a belief in the importance of all three tiers of the Internet of Things: cloud, stratus, and edge. We also emphasize the concept of *services* rather than *devices*. The components of an application are viewed as a collection of interconnected services, and a developer may discover and enlist external services to provide utility functions to their applications. In summary, our system consists of three pieces, a collection of services to manage application execution over the cloud, stratus, and edge tiers, a programming model to help developers construct their application components, and a configuration framework that allows developers to express the interactions between their application components, where and how their application should be executed, and who is authorized to use the application.

2. RELATED WORK

IoT Platforms and Frameworks. The AllSeen Alliance's AllJoyn framework [5] is a prominent IoT framework which aims to create a standard set of protocols over which IoT devices can communicate. AllJoyn is primarily concerned with devices and their interactions, offering a lower level of abstraction than our proposed system. Amazon has recently released its own framework, AWS IoT [2], offering an SDK to integrate connected devices into an ecosystem of AWS-hosted services for messaging and data processing. AWS IoT offers a rich set of features but heavily emphasizes the cloud and ignores the potential of local resources.

Microsoft's HomeOS [9] is focused on device interoperability and facilitating application development for smart homes, but it does not feature a notion of adaptive execution or precise control over deployment. Beam [16] is a framework for inference-based applications featuring an adaptive runtime model in which application modules can migrate, but this migration is based on policies embedded within the system itself that are not exposed to the application developer. The role of local resources in Beam is not clear. Berkeley's Global Data Plane project [18] seeks to incorporate local resources into IoT, but serves as a data storage substrate rather than an application development platform.

Mobile-Cloud Offloading. Our goal of adaptive execution for IoT apps is similar to work that has been done in offloading computations from mobile devices to the cloud, such as MAUI [6]. These systems dynamically migrate computation between mobile phones and a cloud backend to improve performance and conserve battery life. Sapphire [19] expresses mobile-cloud applications as collections of objects representing data and computation, separating application logic from policies on how the application is executed like our system, but it is not concerned with the Internet of Things and its associated challenges.

Distributed Systems and Service Architectures. Many distributed systems have been built around the idea of adaptively migrating data and computation, such as Emerald [13]. The intelligent placement and migration of computation and has also been well studied in the data center literature, e.g. [10], but the Internet of Things involves a much more varied collection of resources, both in terms of hardware capabilities and the desired access control and management policies. Moreover, we are proposing a system that enables more interesting forms of application composition than what is typically seen in the data center. Other systems, such as Ninja [12] and Apache River (Jini) [1], have featured the kind of service-oriented architecture we envision for the Internet of Things, and their ideas merit reconsideration since the advent of smart phones, the cloud, and IoT. Finally, our emphasis on local resources is similar to the notions of cloudlets [15], and our system can be viewed as a means of managing cloudlet resources while also orchestrating application execution on top of them.

Security and Privacy. Much work has been done to facilitate the development of secure applications that leverage the cloud. One prominent example is CryptDB [14], a database that uses homomorphic encryption to perform queries directly over encrypted data, meaning data in the cloud never needs to be decrypted. Unfortunately, CryptDB is still too slow to support general application workloads. We intend to take a different approach by ensuring that sensitive data never reaches the cloud in the first place. Jeeves [17] is a language that allows programmers to control the visibility of sensitive data in different application components. Our proposed system also includes the notion of developer-specified policies, but we focus instead on where to execute application components within a distributed system.

3. MOTIVATION

We motivate our system through a case study object recognition application, depicted in Figure 1. Imagine an augmented reality application intended to support the Internet of Things in which a user can catalog and learn about the smart devices located in a given space by capturing images of these devices on their phone. The object recognition process can be viewed as a pipeline of stages: the application starts with a source image, subsamples the image to reduce its size, selects regions of the image that may potentially contain objects, and then analyzes these regions using a precomputed object model to identify any objects within.

Such an object recognition application has a number of requirements. First, the latency of the recognition process must be low because it is incorporated into the application's interaction with the user, which means response times must be small. Smartphone cameras actually produce images of much higher resolution than what is required for object recognition [8]. However, the collection of images needed to serve as training data for a practical object model is hundreds of gigabytes in size [4], and the model itself requires tens of hours to be computed, even when using high-end GPUs [11]. We may also want to periodically recompute this model as new data becomes available in order to increase its accuracy. Finally, there are privacy concerns regarding the user's images. For example, the user may not want photos of their home to be analyzed in the cloud.

We group the devices comprising the Internet of Things into

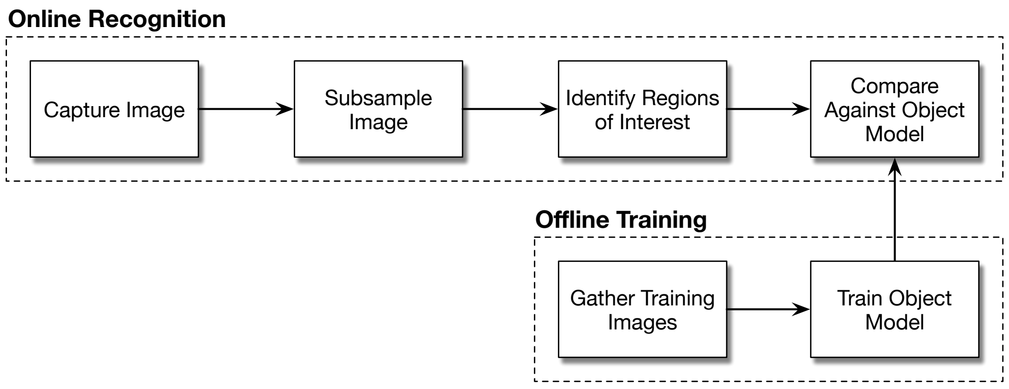

Figure 1: The components of an object recognition application and their relationships.

three tiers. First, there are edge devices consisting of sensors, actuators, embedded microcontroller platforms, and mobile phones. These devices frequently operate with a limited supply of energy, computing power, and network bandwidth, although smartphones now feature reasonably powerful CPUs. The cloud, which forms the opposite end of the spectrum, features practically limitless compute and storage resources but also raises privacy concerns as well as issues of potentially high latency and inconsistent bandwidth. In between these two extremes lies a tier of personal workstations and building servers, which we refer to as the “stratus” layer. This tier has been largely overlooked despite the fact that it offers ample computing resources with low and consistent network latencies in a more trustworthy environment than the cloud.

A new runtime system architecture is needed to enable developers to effectively incorporate all three of these tiers into their IoT applications. Abstractions over heterogeneous hardware must strike the right balance between hiding unnecessary details while exposing the specific capabilities of specific hardware systems. Moreover, the Internet of Things is fundamentally a distributed system rather than a set of client-server interactions, which means we must move beyond the traditional computation offloading techniques. Finally, the IoT setting is characterized by the mobility of both users and devices, which motivates a system that allows applications to adaptively change as they execute. In the remainder of this section, we further motivate our vision and present preliminary microbenchmarks from a prototype object recognition application to examine the potential role of each of these tiers in an IoT runtime system. These microbenchmarks were collected using an LG Mini 2 smartphone running Android 4.4.2, a local server with an Intel Xeon E5-2690 CPU and 64 GB of RAM, and an `m4.xlarge` instance running on Amazon EC2.

3.1 The Stratus Tier

At the heart of our vision is the belief that local resources, like personal workstations, proxies, and gateways, will play an essential role in IoT applications. We envision a system that not only incorporates local resources into its operation but also distributes computation across this tier. Such an approach would be useful to balance load across different machines and to run computation close to the associated data. Moreover, local resources come with a notion of ownership. A user may want to devote resources they control only to applications running on their behalf. Conversely, a user may want computations to run only on machines they trust.

Several of the requirements for object recognition described above can be addressed through effective use of the stratus tier. To achieve low latency and thus fast response times for user interaction, we can perform image analysis against the object model at a local server. Figure 2a shows how response time changes when we move this phase of the recognition process from the cloud to a nearby server. We can see that there is a noticeable difference in latency between the cloud and stratus servers. A second important requirement for our case study is the protection of user privacy by keeping sensitive information off of the cloud, and we can once again use the local tier to address this need. By analyzing images locally, there is no need to send them to the cloud. A potential counterargument to this approach is that images can be encrypted when stored in the cloud, but they must then be decrypted to be processed. Even if we were to use fully homomorphic encryption to operate directly on the encrypted images, this would increase the computational cost by several orders of magnitude [14].

3.2 Edge Processing

In order to incorporate an edge device into a distributed system, the device will require a driver, possibly running on a proxy, to serve as a layer of abstraction over the raw hardware. This implies that distributed edge computation will be an inevitable aspect of any IoT runtime system. We believe that as edge devices become more capable, we can move beyond drivers and begin tasking them with a non-trivial portion of an application’s computational load. This approach is especially advantageous when there is a tradeoff between computation and communication. Edge devices often feature capable CPUs but are still inherently limited by poor or intermittent network connectivity. In our object recognition case study, this motivates subsampling a captured image on the user’s end device rather than at a local or cloud-based server. Subsampling trades increased computation at the device for reduced data transfer over the network. Figure 2b shows that subsampling actually reduces the impact of moving from a local server to a cloud server, demonstrating

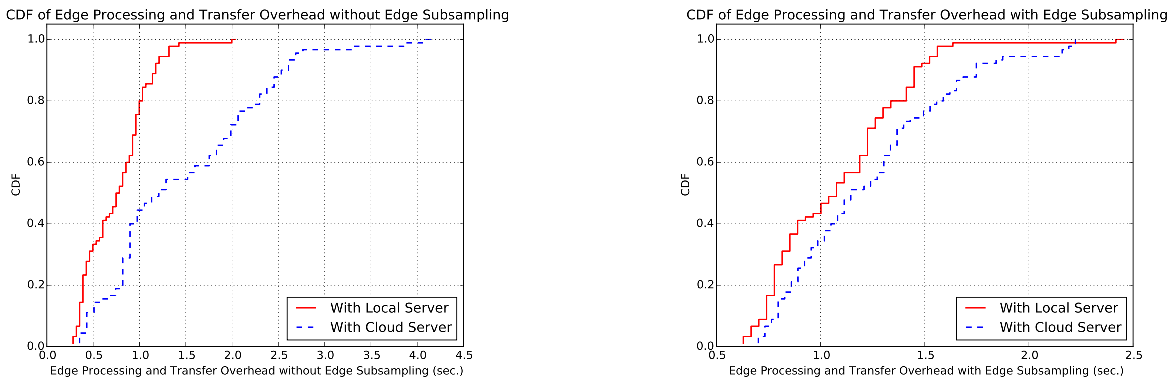

Figure 2: Image subsampling and network transfer time for object recognition hosted at the cloud vs. a local server. We ignore server processing time because of differences in hardware configuration.

an interesting tradeoff in the placement of computation.

3.3 The Role of the Cloud

Although we have advocated for an increased use of local and edge resources in IoT applications, the cloud still has an important part to play. It serves as an excellent setting for compute-intensive tasks and as a repository for sharing data that is not sensitive in nature. In our object recognition application, the object model requires a large body of non-sensitive training data and is expensive to compute, but it only needs to be recomputed periodically. We can therefore avoid storing the training data in the edge or local tiers, where space may be in relatively short supply, and instead keep this information in the cloud. Because model (re)computation is not directly involved in the application's interaction loop, we can perform this asynchronously in the cloud and propagate the object model to the local tier as necessary.

4. SYSTEM DESIGN

We propose a distributed runtime system to facilitate IoT application development, deployment, and management. The discussion of Section 3 informs the design of our system in several ways. First, the system should include resources from all three tiers within the Internet of Things: edge, stratus, and cloud. Application components should be distributed across and within these tiers both to improve performance and to protect sensitive information. As environmental conditions such as the load on a machine or the available network bandwidth change, the system should respond by re-evaluating its placement decisions and possibly migrating application components. However, the developer must be able to express constraints on placement and migration to restrict flows of sensitive information, while stratus and edge resources must be associated with access control policies to prevent unauthorized use of infrastructure.

The Internet of Things, as it stands today, forces developers to reason at a low level about individual devices and nodes, and it suffers from the lack of an analogue to the libraries that have become so indispensable in the development of traditional applications. Therefore, we propose an IoT system that focuses on *services* rather than *devices*, in the vein of a microservice architecture. We define a service as a long-running software module that performs a specific function. It can maintain internal state and asynchronously exchange messages with other services, similar to the actor model of computation. Services act as the building blocks of applications and may range in complexity from a simple device driver to a sophisticated data analysis tool. Our system seeks to foster the sharing of services between developers by allowing applications to incorporate externally-defined services as utilities.

4.1 Service Execution

Figure 3 shows the collection of system components that support execution of user-defined services across the edge, stratus, and cloud tiers. These components are nothing more than special services with additional privileges and constraints on their execution. All services are expected to advertise themselves so they can be discovered by other services and then composed to form full applications. Our system relies on a discovery service that maintains a directory mapping a service's name and metadata to an address where the service can be reached. This service can then be queried to learn about other, new services. The discovery protocol relies on soft state, meaning services must periodically renew their advertisements to remain visible in the directory. Additionally, no special recovery procedure is necessary when restarting a discovery service in the event of a failure.

As shown in Figure 3, each node in the system runs a "resource point" service that allocates and manages the node's resources so that it can serve as an endpoint for user-defined services. A "resource point" service is tasked with monitoring the load on the underlying hardware and performing admission control accordingly. It also includes hardware information as metadata in its advertisement messages to make this available to other services.

Finally, there is the need to intelligently match user services to resource points for execution. This role is filled by a dedicated scheduling service, which requires special algorithms to perform initial placement and adaptively migrate services when environmental conditions change. We expect that each administrative domain of local and edge nodes will run its own scheduling service to control those nodes. This situation is similar to the problem of scheduling tasks to nodes in the data center but involves new challenges. One of the most

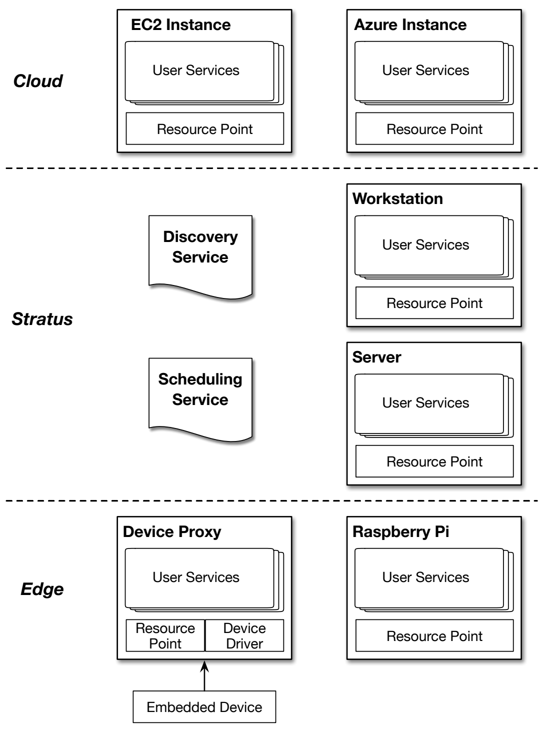

Figure 3: A deployment of our proposed runtime system. Each node runs a resource point service to execute user services. Special discovery and scheduling services run in the local tier. Some edge devices may require proxies to host their drivers.

important challenges is user and device mobility. When a user moves to a new location, it is logical to migrate their supporting applications as well. This can help maintain consistent application performance and ensures that a user can only consume stratus resources in the surrounding area.

4.2 Programming Framework

Not only do IoT developers need a cohesive runtime execution platform, they also need a programming framework that facilitates the rapid development of applications for such a platform. As discussed above, our system expects developers to express their applications as compositions of simple and modular services that communicate with each other via asynchronous message passing. A service will therefore consist primarily of internal state and a set of user-defined callback functions that handle events such as the arrival of a new message and migration of the service to a new node. Along with this programming model, the developer's task can be further eased by a useful set of libraries and primitives. Note, for example, that our programming model is agnostic to the underlying transport technology. Hence, the system can handle the exchange of messages among services internally and is free to use HTTP, a publish-subscribe network, or some other protocol for communication. Similarly, service advertisement can be handled automatically, with the developer free to specify any additional information to include, and discovery by a service's name or metadata is treated as a fundamental system primitive.

4.3 Deployment and Execution Configuration

Once a developer has implemented the services comprising their application, they need to deploy it and specify how the application may be executed on the underlying infrastructure. To deploy an application, the developer must contact the scheduling service that controls the relevant stratus and edge resources, meaning they must know its address in advance. Then, the code for each service is submitted to the scheduler. The scheduler intelligently assigns each service to a resource point. The developer may specify how their application is executed by writing a configuration file that is paired with an application's source code. An application's configuration should identify any external services required by the application. Additionally, the developer should specify the interactions between all services involved in the application, depicted as arrows in Figure 1. The configuration file is where a developer specifies two important classes of constraints: (1) where their services are allowed to run and (2) who is allowed to use these services. To protect sensitive data, some services may be allowed to run only in the edge or stratus tiers. To conserve energy, some services may be excluded from running on edge devices. We might also imagine constraints that involve specific resource points, such as a developer requiring that their service run only on their personal workstation, or a case where a service acts as an abstraction for a particular device and hence is only allowed to run on that device. One of the critical features of this system, then, is to adhere to these constraints in its management of services, allowing the developer to treat them as invariants.

5. DISCUSSION

5.1 Placement and Migration

Application components must be matched to resource points for initial execution and migrated to adapt to environmental changes. How should initial placement decisions be made? One potential approach is to attempt to co-locate computation with its requisite data whenever possible, but how do we know of the relationship between computation and data prior to execution? Should the programmer be expected to enumerate these relationships as part of their application's configuration? Migration brings up similar questions. How should we decide when migration is warranted? What factors into the selection of a destination for migration? Furthermore, migration techniques form a spectrum ranging from immediately transferring all state from source to destination to only transferring data to the destination on demand, when it is actually needed. What points on this spectrum are appropriate for a distributed IoT system like our own?

5.2 Discovery and Scheduling

We have proposed that each local administrative domain contain its own instances of the discovery and scheduling services, but we then must deal with applications that span multiple local domains as well as users and devices that move from one local domain to another. How do we organize the discovery and scheduling services and structure their interactions? For example, how do we handle the transfer of responsibilities from one scheduler to another that accompanies the migration of an application component across local domains? Prior discovery systems have relied on the concept of hierarchy, often based on geographic location [7]. To what extent is this appropriate for our system?

5.3 Security

We mentioned the notion of access control for services above, but how do we effectively enforce access control policies? One could imagine enforcement occurring at several points in the system architecture. This could be done at the discovery level by restricting the visibility of services only to authorized clients, at the transport level by permitting the exchange of messages only between authorized parties, or at the service level by requiring the services themselves to reject unauthorized messages. What are the benefits and drawbacks of each of these choices in the context of IoT application development, deployment, and execution?

5.4 Resource Monitoring and Isolation

How can we monitor computation load and network traffic effectively and with minimal overhead? Once we have this information, how can we leverage it to perform admission control for placement and migration? What tools can we use to attribute resource consumption to specific services that could be deployed in any of the three tiers we have described? Similarly, how do we maintain isolation between services on a shared resource point, and how do we protect the integrity of a host when it is running untrusted code in the form of user services? This has traditionally been the role of virtual machines in the cloud and, more recently, the role of lightweight containers such as Docker [3], but even containers are too cumbersome to deploy on embedded edge devices.

6. CONCLUSION

We have advocated for a new distributed runtime system to facilitate rapid application development for the Internet of Things, informed by the belief that an intermediate tier of resources lying between edge devices and the cloud, is an integral yet underutilized element of the IoT landscape. Our proposed system includes a suite of services to orchestrate the execution of IoT applications, a programming model to simplify the construction of application components, and a configuration engine that allows developers to stipulate where their application's components may run as well as who can access them. We believe that this would represent an important step forward in enabling developers to unleash the potential of the Internet of Things. Our system aims not only to help developers more easily construct applications but also to serve as a platform that offers precise and easily controlled execution semantics for distributed IoT software.

7. REFERENCES

[1] Apache River. `https://river.apache.org/`.

[2] AWS IoT. `https://aws.amazon.com/iot/`.

[3] Docker. `https://www.docker.com/`.

[4] ImageNet Large Scale Visual Recognition Challenge 2014. `http://image-net.org/challenges/LSVRC/2014/index`.

[5] Open Source IoT to advance the Internet of Everything - AllSeen Alliance. `https://allseenalliance.org/`.

[6] E. Cuervo, A. Balasubramanian, D.-K. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, 2010.

[7] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An Architecture for a Secure Service Discovery Service. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 1999.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[9] C. Dixon, R. Mahajan, S. Agarwal, A. J. Brush, B. Lee, S. Saroiu, and P. Bahl. An Operating System for the Home. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, 2012.

[10] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, 2011.

[11] R. Girshick. Fast R-CNN. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015. To Appear.

[12] S. D. Gribble, M. Welsh, R. v. Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, B. Zhao, and R. C. Holte. The Ninja Architecture for Robust Internet-scale Systems and Services. *Comput. Netw.*, 35(4):473–497, Mar. 2001.

[13] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained Mobility in the Emerald System. *ACM Trans. Comput. Syst.*, 6(1):109–133, Feb. 1988.

[14] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011.

[15] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The Case for VM-Based Cloudlets in Mobile Computing. *Pervasive Computing, IEEE*, 8(4), Oct 2009.

[16] R. P. Singh, C. Shen, A. Phanishayee, A. Kansal, and R. Mahajan. A Case for Ending Monolithic Apps for Connected Devices. In *15th Workshop on Hot Topics in Operating Systems*, 2015.

[17] J. Yang, K. Yessenov, and A. Solar-Lezama. A Language for Automatically Enforcing Privacy Policies. In *Proceedings of the 39th Annual ACM Symposium on Principles of Programming Languages*, 2012.

[18] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz. The Cloud is Not Enough: Saving IoT from the Cloud. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, 2015.

[19] I. Zhang, A. Szekeres, D. Van Aken, I. Ackerman, S. D. Gribble, A. Krishnamurthy, and H. M. Levy. Customizable and Extensible Deployment for Mobile/Cloud Applications. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, 2014.