Designing Distribution of Computations for Mobile Cloud Computing Systems

Thesis proposal

YoungHoon Jung
Department of Computer Science
Columbia University
jung@cs.columbia.edu

March 16, 2015

Abstract

A *Mobile Cloud Computing* (MCC) system is a cloud-based system that is accessed by the users through their own mobile devices. MCC systems are emerging as the product of two technology trends: 1) the migration of personal computing from desktop to mobile devices and 2) the growing integration of large-scale computing environments into cloud systems. Designers are developing a variety of new mobile cloud computing systems. Each of these systems is developed with different goals and under the influence of different design constraints, such as high network latency or limited energy supply.

The current MCC systems rely heavily on *Computation Offloading*, which however incurs new problems such as scalability of the cloud, privacy concerns due to storing personal information on the cloud, and high energy consumption on the cloud data centers. I propose to address these problems by answering the following research question: "How computations should be distributed across different computing nodes in MCC systems?"

For a quantitative analysis of mobile cloud computing systems, I have developed the first generation of an innovative design and simulation tool that offers large scalability and heterogeneity support. With this tool system designers and software programmers can efficiently develop, optimize, and validate large-scale, heterogeneous MCC systems.

Leveraging this tool, I developed two new MCC systems where I applied two variations of a new computation distributing technique, called *Reverse Offloading*. By more actively leveraging the computational power on mobile devices, the MCC systems can reduce the total execution times, the burden of concentrated computations on the cloud, and the privacy concerns on storing personal information on the cloud. This approach also creates opportunities for new services by utilizing the information on the mobile instead of accessing the cloud.

I will continue pursuing this line of research by investigating the following ideas: 1) further improving my tool by adding support for modeling mobile GPU systems and 2) developing new MCC applications with new variations of reverse offloading.

The ultimate goal of my research is to enable the design of better mobile applications and cloud-computing platforms. In particular, it will become a vehicle to optimize not only their performance but also their energy dissipation, an aspect of critical importance for any computing system.

This proposal is organized as follows. First, I describe the research problem, the distribution of computations in MCC systems. Second, I present NETSHIP, a design and simulation tool for MCC systems. Third, I describe a cluster of embedded devices and the reverse offloading technique I developed. Fourth, I present a local training MCC application for information extraction and another reverse offloading technique I invented. Fifth, I list related research projects. Finally, I propose a plan to complete my research with a tentative timeline.

Contents

1 Introduction

An increasing number of computing systems rely on a set of backend services operated on cloud computers while providing their primary user interfaces on mobile devices [11, 31, 44]. This new class of emerging computing systems, commonly termed *Mobile Cloud Computing* (MCC) systems, has gained growing popularity in many application domains such as e-commerce [61, 15], learning [73, 56], healthcare [57, 99], gaming [93, 90], social networks [94, 64], and so on. Behind this popularity of MCCs across various domains is the blossoming of two technologies: the wide use of cloud computing and the explosive growth of mobile devices. First, cloud computing has increasingly become the standard way to operate Internet-based services, preferred by the service providers due to its low prices, high performance, and flexibility. These have been the main driving force that increased the instances of cloud servers built in data centers. The number of data centers being built around the world is expected to continue to increase until 2017 when it peaks at 8.6 million [45]. The estimated total space for data centers will reach 1.94 billion square feet in 2018. Second, more and more users access cloud services through their mobile devices which provide a richer experience than using personal computers [30].

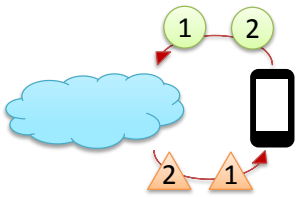
By combining these technologies, MCCs offer many advantages and enable new services. For instance, an MCC speech recognition application takes a segment of the user's voice and sends it to the cloud. The cloud processes the segment and sends the recognition result back to the mobile user. This approach has various advantages compared to running the algorithm on the mobile. First of all, harnessing the powerful processing cores on the cloud allows fast execution of the speech recognition algorithm. Additionally, the database necessary to execute the speech recognition algorithm is stored on the cloud servers, thus freeing space from the limited storage of the mobile device. Meanwhile, the mobile user interface provides a simple yet convenient and ubiquitous way for the users to interact with the application. Lastly, the reduced use of the processing power on the mobile greatly contributes to saving the limited energy budget.

This particular form of task delegation from the mobile to the cloud is called *Computation Offloading*, or simply *Offloading* [63]. Due to its benefits, offloading is very frequently adopted in MCC applications and frameworks [25, 79, 92]. Fig. 1(a) illustrates a mobile device offloading two tasks (Task 1 and Task 2) to the cloud and receiving the corresponding results back from the cloud (Result 1 and Result 2). A variety of efficient offloading techniques have been studied. These techniques are, however, mainly focused on the efficiency of the mobile. For example, some offloading techniques aim to achieve faster total execution time [89], less energy consumption on mobile [18, 29], or both [21, 29]. Eventually, the large amount of offloading translates into more frequent use of the cloud and increased amount of computations from the cloud's perspective.

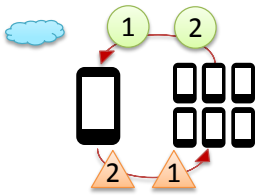
These increased needs started to overburden the cloud, subsequently creating new problems or worsen existing cloud issues.

- The cloud computing services which operate on the data centers currently suffer from scalability. It is difficult to increase the number of server computers hosted in one data center beyond a certain level. Moreover, finding locations and funds to build new data centers is challenging. Behind these challenges, there are some technical constraints like heat management, network, or power supply [36, 37, 91, 17] and social issues such as concerns for the impact on residential environments [83].

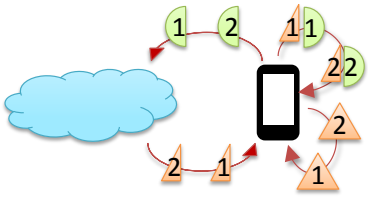
- As a growing amount of personal data is stored on the cloud, users are increasingly concerned



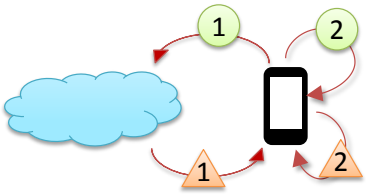
(a) Offloading



(b) Reverse Distributed Offloading



(c) Algorithm-Division Reverse Offloading



(d) Query-Division Reverse Offloading

Figure 1: A Comparison of Offloading and Three Distinct Reverse Offloading Techniques. (A circle is a task and a triangle is the corresponding result.)

about privacy issues. Although it is uncertain whether it is safer to store personal data on the cloud or on the mobile, having the cloud as a data backup, i.e. storing the data on both the mobile and the cloud, might increase the probability that it can be illegally accessed. The recent leaks of celebrity photos in 2014 [81, 82] is the evidence that the security issue on the cloud system can turn into a large-scale threat for personal privacy.

- The energy consumed by data centers is enormous. Data centers are the largest and fastest growing electricity consumer. US data centers consumed an approximately 91 billion kilowatt-hours of electricity in 2013. This amount was twice as much as the power consumed by all the households in New York City the same year. The energy consumption by data centers is anticipated to reach 140 billion kilowatt-hours by 2020 [62]. The companies that operate these large-scale data centers are under pressure to reduce their energy consumption from governments and environmental organizations.

The fundamental research question behind these problems is **how to optimally distribute computations across different computing nodes** in the systems. The question is very important particularly for MCC systems where two very distinct types of computers must cooperate. For instance, preprocessing the data on the mobile before transferring them to the cloud may substantially decrease the amount of data the cloud must process as well as the network use. This is a well-known question studied by many researchers who have proposed different solutions optimized to achieve different objectives. Examples include offloading for execution times based on resource monitoring [97], migrating tasks to the cloud for saving energy [58], distributing application binaries automatically [42], and partitioning the application dynamically [20].

However, these approaches are not suitable for the increasing number of MCC applications due to the following limitations:

- System optimization is application specific. Different applications benefit from different heuristics that can largely save the total amount of computations required by the applications.

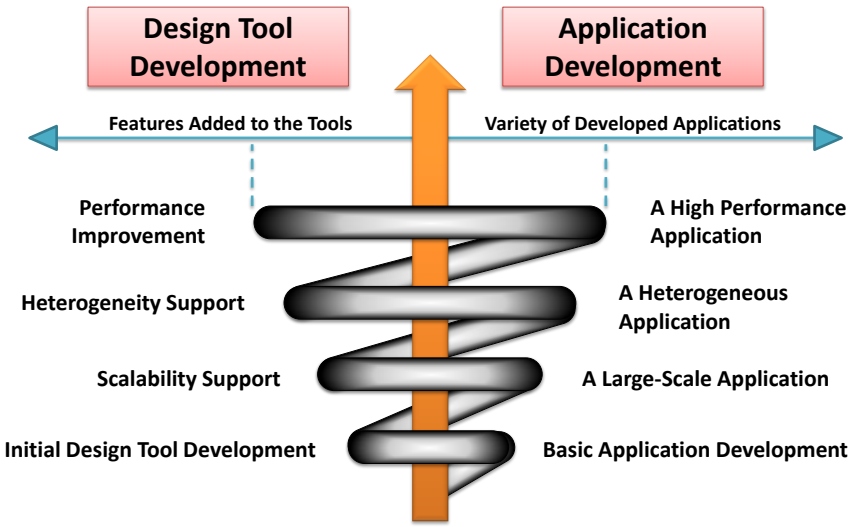


Figure 2: A Cooperative Development Process of Design Tools and Applications.

An automated analysis cannot invent a new technique to optimize the application. Instead, a system designer with a fast and effective design tool is more likely to come up with an efficient optimization idea through design-space exploration.

- Existing methods cannot support heterogeneity and very large scalability, the two most distinguished characteristics of MCC systems. MCC systems are heterogeneous because they consist of cloud computers and mobile devices. Meanwhile, those mobile devices have varying computing processors and networks. In addition, recent large-scale MCC systems serve more than a billion users with more than tens of millions of them accessing the cloud at the same moment.

- Computation offloading is a viable model only under the circumstance where the growth of cloud computers exceeds the growth of mobile devices. Therefore, offloading is not applicable to many MCC systems where mobile devices may outgrow the available cloud resources. Furthermore, computation offloading is increasingly used also in other types of fast-growing computing systems such as Internet of Things [34] and Cloud Robotics [53]. Hence, the computational burden on the cloud is expected to intensify further in near future.

I propose to study **the optimal distribution of computations in the MCC systems** by

1. *developing effective and efficient design and simulation tools to support the design and optimization of large-scale heterogeneous MCC systems* and

2. *inventing a class of patterns for computation distribution and apply them to the new MCC systems I develop using my tools.*

The way I plan to study the distribution of computations for MCC systems is an iterative process of developing design tools and implementing MCC systems where distinct distributions of computations can be applied. The development of design tools and applications can benefit each other, accelerating the completion of both. As shown in Fig. 2, these two development processes are iteratively connected. For instance, a simulation and design tool makes the development and

testing of a new application more efficient. To assist the developers of a new type of application that runs a large number of distributed mobile devices, the tool should support great scalability to simulate many nodes. When the application runs on heterogeneous cores, the tool also needs to support these distinct types of cores. Without an appropriate design tool, application development can be slow. On the other hand, without knowing actual applications, the tools cannot be enhanced in a meaningful direction. Hence, the development process of the design tools and the applications is mutually beneficial.

Throughout the development of various MCC applications, I invented a series of techniques called *Reverse Offloading*. Reverse offloading distributes computations across the cloud and the mobile devices, instead of offloading the entire tasks to the cloud. Fig. 1(b)~ 1(d) illustrates three different types of reverse offloading. First, *Reverse Distributed Offloading* distributes the entire tasks across multiple mobile devices in the same network. This makes the mobile independent from the cloud, as shown in Fig. 1(b). In other words, the cluster formed out of the mobile devices works like an alternative cloud. Second, *Algorithm-Division Reverse Offloading* splits each task into two portions. As shown in Fig. 1(c), the application sends a portion of the task to the cloud; then, the intermediate result for the portion of the task returns to the mobile; finally, the mobile device runs the rest of the task taking as input the intermediate result returned from the cloud. Third, *Query-Division Reverse Offloading* selectively sends only some of the tasks to the cloud. I discussed these 3 techniques in the context of three MCC applications in Section 3, 4, and 6.2, respectively.

Reverse offloading can effectively address the aforementioned challenges as follows:

- It reduces the amount of computations that the cloud must run. Reverse distributed offloading makes the mobile devices independent from the cloud. In other words, the computational burden on the cloud is dissolved. MCC applications that adapt algorithm-division reverse offloading will only offload certain portions of the tasks to the cloud. In this case, the rest is saved from offloading. Query-division reverse offloading sends only certain tasks to the cloud. Likewise, the tasks handled by the mobile devices are saved from offloading. Particularly, the amount of computations reduced by reverse offloading is proportional to the number of mobile devices, which is the most rapidly growing factor in the MCC systems. The fast improving computational power and energy efficiency of mobile devices also contribute to the effectiveness of reverse offloading.

- Reverse offloading can effectively decrease the total execution time for some applications.This is due to the reduced use of network, particularly on the mobile side. The mobile network, i.e. Wi-Fi or LTE, takes a large portion of the execution time and the energy consumption of MCC applications. By reducing the network use, reverse offloading inherently decreases the network latency and energy dissipation. Also, the growing processing power on mobile, e.g. mobile CPUs and GPUs, contributes significantly to the feasibility of reverse offloading.

- A certain type of reverse offloading can mitigate some privacy concerns. Instead of sending personal information to the cloud (while also leaving it on the mobile), securing it only on the mobile provides less chance for the personal information to be leaked.

- Reverse offloading can enable new services. For instance, in Section 4 I introduce an MCC application that uses personal information in the users' emails by extracting the information

from the messages stored on the mobile devices. So far this type of services has been possible only for a small group of large companies that offer their own email services, e.g. Google, Microsoft, and Yahoo. Instead of accessing email services on the cloud, leveraging the personal information in the email stored on the mobile makes a variety of new services feasible for many small independent companies.

The rest of the proposal is organized as follows. In Section 2, I present a design tool for developing MCC applications. Particularly, this tool supports large scalability and heterogeneity required by the MCC systems. This tool, named NETSHIP, was presented at DAC'13 [49]. In Section 3 I introduce the reverse distributed offloading adapted on a cluster of embedded systems. The material in this section was published at CloudCom'12 [48]. In Section 4, I introduce an MCC application where algorithm-division reverse offloading is applied. Specifically, this study shows that the execution can be faster when the mobile processing core is more actively used, mainly due to the reduced amount of network use by reverse offloading. The material in this section will be presented at WWW'15 [50]. In Section 5, some related projects are reviewed. A plan to complete my research is given in Section 6.

2 A Design Tool For Heterogeneous Large-Scale MCC Systems

From a single SoC to a network of embedded devices communicating with a backend cloud-computing server, emerging classes of embedded systems feature an increasing number of heterogeneous components that operate concurrently in a distributed environment. As the scale and complexity of these systems continues to grow, there is a critical need for scalable and efficient simulators. Together with Jihyung Park, Michele Petracca, and Luca Carloni, I developed a *Networked Virtual Platform* as a scalable environment for modeling and simulation. The goal is to support the development and optimization of embedded computing applications by handling heterogeneity at the chip, node, and network level. To illustrate the properties of our approach, I presented two very different case studies: the design of an Open MPI scheduler for a heterogeneous distributed embedded system and the development of an application for crowd estimation through the analysis of pictures uploaded from mobile phones.

2.1 Introduction

Computing systems are becoming increasingly more concurrent, heterogeneous, and interconnected. This trend happens at all scales: from multi-core systems-on-chip (SoC), which host a variety of processor core and specialized accelerators, to large-scale data-center systems, which feature racks of blades with general purpose processors, graphics-processor units (GPUs) and even accelerator boards based on FPGA technology. Furthermore, nowadays many embedded devices operate while being connected to one or more networks: e.g., modern video-game consoles rely on the Ethernet protocol [80], millions of TVs and set-top boxes are connected through DOCSIS networks [40], and most smartphones can access a variety of networks including 3G, 4G, LTE, and WLAN [54, 46, 88].

As a consequence, a growing number of software applications involve computations that run concurrently on embedded devices and backend servers, which communicate through heterogeneous wireless and/or wired networks. For example, *mobile visual search* is a class of applications

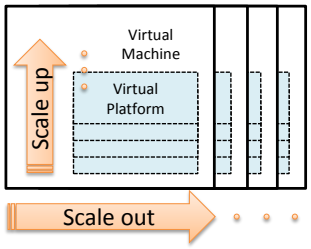


Figure 3: The two orthogonal scalabilities of NETSHIP.

which leverages both the powerful computation capabilities of smart phones as well as their access to broadband wireless networks to connect to cloud-computing systems [33, 85].

We argue that the design and programming of these systems offer many new unique opportunities for the electronic design automation (EDA) community. For instance, system and sub-system architects need tools to model, simulate, and optimize the interaction of many heterogeneous devices; hardware designers need tools to characterize the applications, software and network stack that they must support; and software developers need early high-level modeling environments of the underlying hardware architecture, often much before all its components are finalized.

As a step in this direction, we present NETSHIP, a networked virtual platform to develop simulatable models of large-scale heterogeneous systems and support the programming of embedded applications running on them. Users of NETSHIP can model their target systems by combining multiple different virtual platforms with the help of an infrastructure that facilities their interconnection, synchronization, and management across different virtual machines.

Given a target system, NETSHIP can be used to set up a simulation environment where each VP works as single-device simulator running a real software stack, e.g. the Linux operating system, with drivers and applications. Thus, it makes it possible to run real applications over the entire distributed system, without actually deploying the devices. This allows users both to jump start the functional verification process of the software and to drive the design optimization process of the hardware and the network.

While in certain areas the terms *virtual platform (VP)* and *virtual machine (VM)* are often used without a clear distinction, in our research it is particularly important to distinguish them. A VP is a simulatable model of a system that includes processors and peripherals and uses binary translation to simulate the target binary code on top of a host instruction-set architecture (ISA). VPs enable system-level co-simulation of the hardware and the software parts of a given system before the actual hardware implementation is finalized. Instead, a VM is the management and provisioning of physical resources in order to create a virtualized environment. The resources are mostly provided by one or more server computers and the management is performed by a *hypervisor.* Examples of VPs include OVP, VSP, and QEMU, while KVM, VMware, and the instances enabled by the Xen hypervisor are examples of VMs. [1]

Thanks to its novel *VP-on-VM model*, the NETSHIP infrastructure simplifies the difficult process of modeling a system with multiple different VPs. In fact, the ability to support multiple VPs interconnected through a network makes NETSHIP free from the limitation of one specific VP while providing access to the superset of their features. For example, users who are interested in modeling an application running in part on certain ARM-based mobile phones and in part on MIPS-based servers can use NETSHIP to build a network of Android emulators [2] and OVP nodes.

[1] Recent efforts to run VMs on embedded cores [22, 59] remain within the VM definition as they do not adopt binary translation.

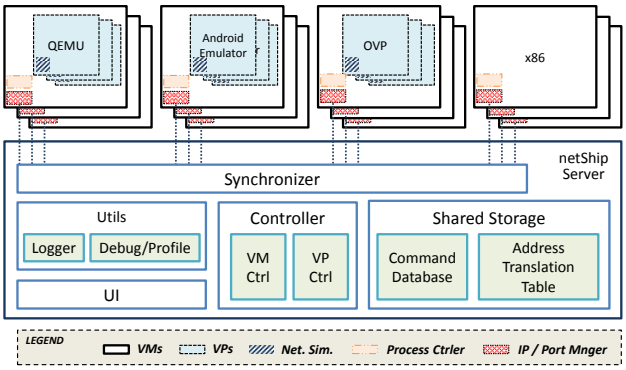


Figure 4: The architecture of NETSHIP.

The VP-on-VM model makes NETSHIP scalable both horizontally and vertically, as illustrated in Fig. 3. The users can scale the system out by adding more VM instances to the network (horizontal scalability) and scale the system up by assigning to each VM instance more CPU cores on which more VP instances can run (vertical scalability).

Another pivotal advantage the VP-on-VM model adds to NETSHIP is access to the features of VMs, i.e. pausing, resuming the VM instances, duplicating instanced preconfigured for specific VP types, or migrating them across physical machines.

Contributions. The main goal of this research work is to understand how to build and use a *Networked Virtual Platform* for the analysis of distributed heterogeneous embedded systems. To do so, we built NETSHIP as a prototype based on the VP-over-VM model with the main objectives of supporting heterogeneity and scalability. To the best of our knowledge, this is the first work that presents this type of CAD tool. To evaluate NETSHIP we have completed a series of experiments including two complete case studies. The first case study shows how a networked virtual platform can be used to better utilize the computational resources that are available in the target system while guaranteeing certain performance metrics. The second case study shows how a networked virtual platform can be used to develop a software application running on a heterogeneous distributed system that consists of many personal mobile devices and multiple computer servers while, at the same time, obtaining an estimation of the resource utilization of the entire system.

2.2 Networked Virtual Platforms

A heterogeneous distributed embedded system can consists of a network connecting a variety of different components. In our approach, we consider three main types of heterogeneity: first, we are interested in modeling systems that combine computing nodes based on different types of processor cores supporting different ISAs (*core-level heterogeneity*); second, nodes that are based on the same processor core may differ for the configuration of hardware accelerators, specialized coprocessors like GPUs, and other peripherals (*node-level heterogeneity*); third, the network itself can be heterogeneous, e.g. some nodes may communicate via a particular wireless standard, like GSM or Wi-Fi, while others may communicate through Ethernet (*network-level heterogeneity*.)

NETSHIP provides the infrastructure to connect multiple VPs in order to create a networked VP that can be used to model one particular system architecture having one or more of these heterogeneity levels. For example, Fig. 4 shows one particular instance of NETSHIP which is obtained by connecting multiple instances of the QEMU machine emulator [14], the Android mobile-device

emulator [2], and the Open Virtual Platform (OVP) [9].

Each VP instance runs an operating system, e.g. Linux, with all the required device drivers for the available peripherals and accelerators. The application software is executed on top of the operating system. Each VP typically supports the modeling of a different subset of peripherals: e.g., OVP supports various methods to model the hardware accelerators of an SoC: users can write models in SystemC TLM 2.0 or take advantage of the BHM (Behavioral Hardware Modeling) and PPM (Peripheral Programming Model), which are C-compatible Application Programming Interfaces (APIs) that can be compiled using the OVP-supplied PSE tool-chain[2].

In addition to the features supported by each particular VP, we equipped NETSHIP with all the necessary instrumentation to: (1) enable multiple instance executions; (2) configure port forwarding; and (3) measure the internal simulation time. Furthermore, any node in the network of VPs could potentially be a real platform, instead of being a virtual one: e.g. in Fig. 4, each of the x86 processors runs native binary code and still behaves as a node of the network.

One of the main novelty aspects of NETSHIP is the *VP-on-VM model* which is critical for the scalability of modeling and simulations. We designed NETSHIP so that multiple VP instances (e.g., 2 to 8) can be hosted by the same VM. By adding more VMs, the number of VPs in the system can be increased with a small performance penalty, as discussed in Section 2.3. Notice that the simple action of cloning a VM image that includes several VPs often represents a convenient way to scale out the model of the target system.

Next, we describe the main building blocks of NETSHIP.

Synchronizer. VPs vary in the degree of accuracy of the timing models for the CPU performance that they support. Some VPs do not have any timing model and simply execute the binary code as fast as possible. This is often desirable, particularly when a VP runs in isolation. In NETSHIP, however, we are running multiple VPs on the same VM and, therefore, we must prevent a VP from taking too much CPU resources and starving other VPs. QEMU provides a crude way to keep simulation time within a few seconds of realtime. OVP, instead, controls the execution speed so that the simulated time never surpasses the wall clock time. Multiple OVP instances, however, still show different time developments which require a synchronization method across the VPs in the network.

We equipped NETSHIP with a *synchronizer* module to support synchronization across the heterogeneous set of VPs in the networked platform, as shown in Fig. 4. The synchronizer is a single process that runs on just one particular VM and is designed in a way similar to the fixed-time step synchronization method presented in [23]: at each iteration, a central node increases the base timestamp and the client nodes stop after reaching the given timestamp. However, we considered two aspects in our synchronizer:

- we must synchronize VPs that might be scattered over several physically-separated machines;
- we must preserve the scalability provided by the VP-on-VM model.

NETSHIP targets large-scale systems which involve deployments across physically- separated machines where millisecond-level network packet travelling is actually required to synchronize. Hence, NETSHIP supports the modeling of applications that have running times ranging from a few seconds to multiple hours or days, rather than simulations at nanosecond-level.

[2]PSE is Imperas Peripheral Simulation Engine [9].

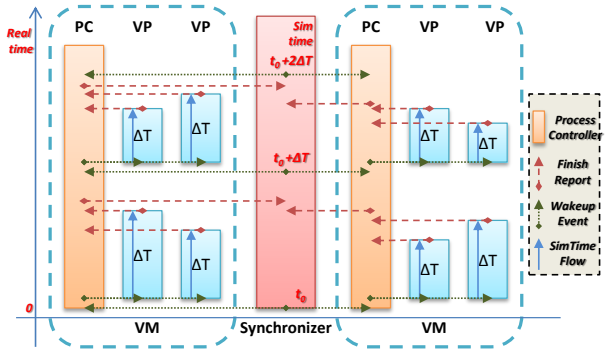
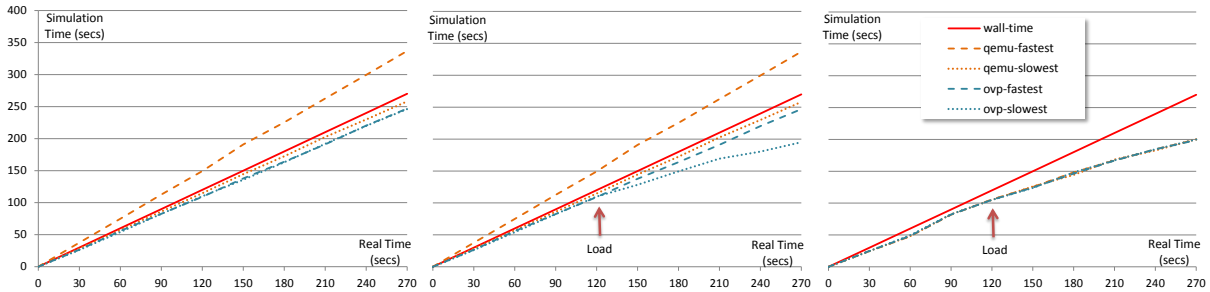


Figure 5: Synchronization process example.



(a) No load, No sync (b) Load, No sync (c) Load, Sync

Figure 6: Simulation time measurements.

To support synchronization over the VP-on-VM model, we designed a *Process Controller* (PC) that allows us to manage the VPs in a hierarchical manner. Each VM hosts one PC, which controls all the VPs on that VM. In particular, all messages sent by a VP to the synchronizer pass through the PC. The PC supports also running programs on a host machine: e.g. in the case of Fig. 4, the PCs manage the synchronization of the processes running on a x86 through the two POSIX signals `SIGSTOP` and `SIGCONT`, in the same way as the UNIX command `cpulimit` limits the CPU usage of a process.

Fig. 5 illustrates an example of the synchronization process with two VMs, each hosting two VP instances. The following steps happen at each given iteration i:

1. the synchronizer issues a future simulation time $t_i = t_{i-1} + \Delta T$ to the VPs and wakes them up;
2. the VPs run until they reach the appointed time t_i and report to their PC;
3. As soon as a PC receives reports from all the connected VPs, it reports to the synchronizer;
4. After the synchronizer has received the reports from all the PCs, it loops back to Step 1.

The users can configure the time step ΔT to adjust the trade-off between the accuracy and the simulation speed.

Let's compare the complexity of this hierarchical method to the existing method. In the synchronization algorithm in [23], if the number of VP is $|VP|$ the synchronization process should receive and count $|VP|$ reports to make sure that all the VPs have reached to the appointed simulation time. This results in a $\Theta(|VP|)$ algorithm complexity in *Synchronizer*, whereas in NETSHIP it is $\Theta(\sqrt{|VP|})$ because *Synchronizer* manages $\sqrt{|VP|}$ PCs, each of which controls $\sqrt{|VP|}$ VPs.[3]

[3]It may be enough for *Synchronizer* only to count the number of reports from PC to know that every VP instance is ready and advance the

Command Database. NETSHIP was designed to support the modeling of systems with a large scale of target networked VPs. In these cases, to manually manage many VP instances becomes a demanding effort involving many tasks, including: add/remove new VP instances to/from a system, start the execution of applications in every instance, and modify configuration files in the local storage of each instance. In order to simplify the management of the networked VP as a whole, we developed the *Command Database* that stores the script programs used by the different NETSHIP modules. For example, the network simulation module and IP/Port forwarding module load the corresponding scripts from the database and execute them. Table 1 contains a detailed list of the commands in the database.

Name	Behavior
vp_ctrl_pwr	turns the VP on/off
net_set_bw	sets the VP's network bandwidth to simulate
net_set_delay	sets the VP's network delay to simulate
net_set_error	sets the VP's network error rate to simulate
net_load_rt	loads the address/port settings to use
cmd_execute	executes a command in all the VPs
acc_gen	loads driver modules and creates a device node for the specified accelerator
report_local	reports the local time in the VP
report_cpu	reports the cpu time in the VP

Table 1: List of commands in the command database.

VM and VP Management. Whereas the commands in the Command Database are dedicated to VP configuration, we developed specialized modules to manage the VPs and the VMs (for the latter we integrated tools provided by the VM vendor). These modules manage the disk images of the VMs and VPs, for creating, copying, and deleting their instances. Since many VPs are still in the early stages of development and are frequently updated by the vendors, the VP management module checks the availability of new updates for all the installed VPs.

Network Simulation. The VP models of NETSHIP are provided with their own models of the network interface card (NIC). These models, however, are purely behavioral and do not capture any network performance property, such as bandwidth or latency [23]. Consequently, we developed a *Network Simulation* module that enables the specification of bandwidth, latency, and error rates, thus supporting the modeling of network-level heterogeneity in any system modeled with NETSHIP. As shown in Fig. 4, a Network Simulation module resides in each particular VP and uses the traffic-shaping features based on the *tc* command, which manipulates the traffic control settings of the Linux kernel.

Address Translation Table. In NETSHIP there are two points where packet forwarding plays a critical role:

1. To allow incoming connections to the VPs through their emulated NIC model, most VPs provide a way to redirect a port of the host to a port of the VP, so that packets that arrive to that VM port are redirected to the corresponding VP port. We leverage this redirection

simulation time. However, this method is unreliable in the sense that there is no way for *Synchronizer* to tolerate a PC malfunctioning. If a hash table, for example, is used to map a PC's IP to the data structure for checking that the PC is reporting more than once in a cycle, the average complexity of the algorithm in [23] is $O(|VP| + |VP|^2/k)$ and for our algorithm it is $O(\sqrt{|VP|} + \frac{|VP|}{k})$, where k is the number of buckets in the hash table and searching n times in a hash table takes $n * O(1 + n/k)$.

Library	Option	Default Value
SSH (fixed)	Port	22
Hadoop (fixed)	dfs.http.address	50070
	dfs.datanode.http.address	50075
	mapred.job.tracker.http.address	50030
	mapred.task.tracker.http.addres	50060
Open MPI (random)	oob_tcp_port_min_v4	0
	oob_tcp_port_range_v4	65535
	btl_tcp_port_min_v4	1025
	btl_tcp_port_range_v4	65525

Table 2: Example of library port uses.

mechanism so that the applications running on the VPs can open ports to receive packets from other VPs, even if those are located on a physically separated VM. [4]

Port forwarding is the technique of redirecting the traffic incoming on one network port of the OS running on the host VM towards a specific port of the OS running on the hosted VP. For example, when a packet arrives to Port 10020 of the VM's OS, the VP to which Port 10020 is assigned intercepts the packet and forwards it to Port 22 of the VP's OS. Hence, when users connects through SSH to the host's IP and Port 10020 they are forwarded to Port 22 of the VP. This is configured in the behavioral model of the VP and performed through the NIC model.

Unlike SSH, some libraries require a random port to be accessed by clients; for instance Open MPI communicates through random ports ranging from 1025 to 65535 [35]. However, most libraries also provide a way to change or reduce the required port range as shown in Table 2. We reduced the range and mapped it to the same port range on the virtual addresses, 200.0.0.x. One of these addresses is allocated to each of VP instances using *iptables* through the Port Management module in Fig. 4.

2. Since certain applications required that each VP must be accessible through a unique IP address and generally there is only one physical IP address per VM, we must map each VP to a virtual IP address. Each VP must know such mapping for all other VPs in the system. Hence, we used the UNIX command `iptables` to create a table of assignments within the kernel of each VP. NETSHIP stores the translation information in the *Address Translation Table*, which is loaded through the network commands stored in the Command Database.

2.3 Scalability Evaluation

In this section, we experiment NETSHIP from the synchronization, scalability, performance, and network-fairness perspectives.

Simulated Time and Synchronization. Eight OVP instances and eight QEMU instances are running in this simulation setup. The three figures in Fig. 6 show the *simulated time* in each. The red solid line represents the time graph of an ideal VP, with $y = x$, where y is the *wall-clock time* and x is the simulated time. While there were multiple instances running together, in the figure we

[4] While certain VPs provides a network bridge feature that allows more generic network functionalities, we use port redirection because it is commonly supported by every VP family.

VP Type	Core Model	CPU use	Preferred #VPs
OVP	Accelerator	~ 24%	4
OVP	MIPS	~ 6%	16
QEMU	PowerPC	~ 12%	8
VMWare	x86	~ 5%	20

Table 3: Host CPU use of each VP.

show only the fastest and slowest instances for each VP family, in order to summarize the range of variations within each VP family and to better compare the VP families.

Fig. 6(a) measures the simulated time of unloaded VPs. Each VP advances its simulated time linearly, but differently from each other. In particular, the range of simulated time among QEMU instances is wide: from 4% slower up to 25% faster than the wall-clock time. Instead, the OVP instances show almost the same simulation speed (0.3% variation), which is 8% slower than the slowest QEMU instance. This reflects the fact that OVP has a better method to control the simulation speed.

Fig. 6(b) shows the case when a VP is subject to a heavy workload. In particular, at simulated time $x = 120s$ one OVP instance starts using a high-performance accelerator. From that point on, the OVP instance gets slower than every other instances, as shown by the deviation among the OVP lines in the figure. This is natural when the peripherals are modeled at a very high level of abstraction. In a fair host VM, all VPs are granted the same amount of CPU time to be executed. Simulating the use of a hardware accelerator on a VP typically requires the VP process on the VM to executes a non-negligible computation. In the other words, running the functional model of the accelerator uses the VM's CPU resources and requires a certain amount of wall-clock time. From the viewpoint of simulated time, however, this computation happens in a short period of time (due to the accelerator's timing model); therefore the given VP instance becomes slower than the others. The misalignment of the simulated time among VP instances is a concern when simulating distributed systems, because it might cause the simulated behaviors to be not representative of reality.

To address this problem, we implemented the synchronization mechanism explained in Section 2.2. Fig. 6(c) shows the behavior of all VP instances under the same conditions but with the synchronization mechanism turned on (with a synchronization cycle of 300ms). The simulated time of all VPs becomes the same as the slowest instance. The synchronization cycle can be decided by the users. Our experiments show that it should not be too small ($\geq$ 1ms) because: i) a synchronization that is much more frequent than the OS scheduling time slice[5] may disturb the timely execution of the VPs, and ii) the synchronization is an overhead and slows down the overall simulation.

Vertical Scalability. By *vertical scalability* we mean the behavior of the networked VP as more VPs are added to a single VM. As discussed above, although the synchronizer preserves the simultaneity of the simulation among VPs, it makes them all run at the speed of the slowest instance , i.e. even one slow VP instance is enough to degrade the simulation performance of the whole system. Therefore, an excessive number of VP instances on the same VM will likely cause a simulation slowdown.

Table 3 shows the amount of CPU of the host VM that is used by a VP instance. For example,

[5]Linux O(1) scheduler dynamically determines the time slice, ranging from milliseconds to a few hundreds milliseconds.

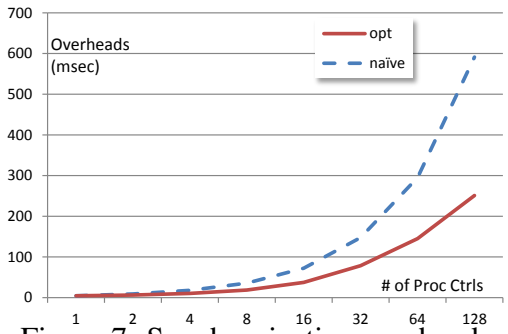


Figure 7: Synchronization overheads.

when an OVP instance fully utilizes one accelerator, it takes up to 24% of the host CPU resource in the hosting VM. This means that 4 is the optimal number of OVP instances, equipped with that accelerator, which can co-exist on the same VM without performance penalty. Likewise, the CPU of a QEMU PowerPC that is fully busy, i.e. a simulated 100% utilization, uses up to 12% of the host VM's CPU resources: hosting up to 8 QEMU PowerPC instances in the same VM is performance optimal.

Note that even if the number of VP instances goes over the optimal value, the synchronizer still preserves the simultaneous simulation of all nodes. However, balancing out the number of VP instances hosted across the VPs, or alternatively increasing the computational resources available to the VM, helps to increase the overall simulation performance.[6]

Synchronization Overheads and Horizontal Scalability. *Horizontal scalability* describes the behavior of the simulated VP as we scale the number of VMs. The synchronizer is the entity in the networked VP that communicates with all VMs in order to keep all VPs aligned. Fig. 7 shows the overhead increase as the number of VMs grows. We measured the overhead as the time elapsed from when the slowest VP instance reports to have terminated the execution step to the time the same instance starts the new one. We experimented with ten VP instances insisting on each PC. In the figure we compare a naïve implementation with an optimized implementation of the synchronizer. For both version the principle of the synchronization is the same; however, in the optimized version we used more advanced techniques to reduce the communication latency and overhead, using the following methods:

Multicasting-based Wakeup. In order to reduce the serial latency of the wakeup packets delivered from the synchronizer to the PCs, we used multicast UDP.

Atomic Operations in Shared Memory for In-Machine Reporting. The PC must check that VPs correctly report the end of the current simulation cycle. This is done by having each VP increase a shared counter through an atomic operation. This is possible because all VPs are on the same machine.

Disabling Nagle's Algorithm. Unlike the waking-up message of the synchronizer to the PCs (1-to-N), multicast UDP cannot be used to carry reports from the PCs to the synchronizer (N-to-1). In the Linux kernel, TCP sockets typically use by default an optimization technique, Nagle's algorithm, which combines a number of small outgoing packets and sends them all in one single message [66]. This method, however, increases the latencies of these small packets (up to 30ms in our experiments), which is a critical issue in our synchronization design, since latency is way

[6]The CPU resources of the VM might not be the only bottleneck. For a more generic approach, an analysis of disks, network congestion, memory bandwidth, bus capacity, and cache interference are required. In our experiments, however, the constraint due to the VM's processing power was the most dominating factor that decides Vertical Scalability.

more important than throughput. We then disabled the Nagle's algorithm by turning on the socket option TCP_NODELAY for each TCP socket.

Using POSIX Signals to Sleep and Wake up. In order to stop and wake-up a VP instance our PC uses two signals: *SIGSTOP* and *SIGCONT*. The use of standard Linux signals provides several advantages. First, the PC can be easily implemented in a separate user space program, without the knowledge of the internals of the VPs. Second, once implemented, the PC is portable across the VPs, requiring no modifications. Third, the PC can stop all threads in the process, while sleeping works only for the thread of the current context. Most importantly, this also enables a synchronized execution with processes that run natively on a host VM, e.g. x86 server, outside of any VP.[7]

The overhead for 128 PCs is approximately $250ms$, which slows down the networked VP by about 25% if the simulation step is set to $1s$.

Although the optimized implementation significantly reduces the overhead, both slopes increase linearly with respect to the number of PCs in the network (notice that the x-axis is logarithmic). This is because synchronization involves all PCs, each of which is located in separate machine, and all reports require a packet transmission across the network and linear-time computation to parse the reports.

In summary, the synchronization across VMs limits the horizontal scalability, in the sense that the simulation step after which all VPs are synchronized, must be (much) bigger than the time it takes to actually perform the synchronization, which strongly depends on the characteristics of the hosting VMs and how they are connected.

2.4 Summary

We have designed and implemented NETSHIP, a framework for building networked VPs that model heterogeneous distributed embedded systems. Networked VPs can be utilized for various purposes, including: i) simulation of distributed applications, ii) systems, power, and performance analysis, and iii) costs modeling and analysis of embedded networks' characteristics.

We also designed hardware accelerators for specific algorithms. We analyzed that accelerators might require more resources of the CPUs that host the simulation. We quantified how this phenomenon partially limits the scalability of the entire networked VP, and provided guidelines on how to distribute the VPs in order to counter balance this loss of simulation performance.

Finally, we used NETSHIP to develop two networked VPs. We used one VP to design a scheduler based on MPI and to verify through simulation how the scheduler is able to optimize the execution of many MPI jobs over a network of heterogeneous machines, by simply distributing the jobs among the available machines on the basis of their performance-per-application profile. We used the other VP to design and validate an application distributed among portable devices and a cloud of servers, and also to derive potential insight about the number of servers and the image size that guarantee the entire application to run in real-time.

[7] In NETSHIP x86 binaries are executed on a VM, not a VP. Through the stop and continue signals PC synchronizes the process without modifying the binary executables.

3 Reverse Distributed Offloading: A Cluster Of Embedded Systems

An expanding wealth of ubiquitous, heterogeneous, and interconnected embedded devices is behind most of the exponential growth of the "Big Data" phenomenon. Meanwhile, the same embedded devices continue to improve in terms of computational capabilities, thus closing the gap with more traditional computers. Motivated by these trends, together with Richard Neill and Luca Carloni, I developed a heterogeneous computing system for MapReduce applications that couples cloud computing with distributed embedded computing. Specifically, our system combines a central cluster of Linux servers with a broadband network of embedded set-top box (STB) devices. The MapReduce platform is based on the Hadoop software framework, which we modified and optimized for execution on the STBs. Experimental results confirm that this type of heterogeneous computing system can offer a scalable and energy-efficient platform for the processing of large-scale data-intensive applications.

3.1 Introduction

The growth in the amount of data created, distributed and consumed continues to expand at exponential rates: according to a recent research report from the International Data Corporation, the amount of digital information created and replicated has exceeded the zettabyte barrier in 2010 and this trend is expected to continue to grow "as more and more embedded systems pump their bits into the digital cosmos" [43]. In recent years the MapReduce framework has emerged as one of the most widely used parallel computing platforms for processing data on very large scales [52]. While MapReduce was originally developed at Google [24], open-source implementations such as Hadoop [3] are now gaining widespread acceptance.

The ability to manage and process data-intensive applications using MapReduce systems such as Hadoop has spurred research in server technologies and new forms of Cloud services such as those available from Yahoo, Google, and Amazon.

Meanwhile, the Information Technology industry is experiencing two major trends. On one hand, computation is moving away from traditional desktop and department-level computer centers towards an infrastructural core that consists of many large and distributed data centers with high-performance computer servers and data storage devices, virtualized and available as Cloud services. These large-scale centers provide all sorts of computational services to a multiplicity of peripheral clients, through various interconnection networks. On the other hand, the increasing majority of these clients consist of a growing variety of embedded devices, such as smart phones, tablet computers and television set-top boxes (STB), whose capabilities continue to improve while also providing data locality associated to data-intensive application processing of interest [67, 68]. Indeed, the massive scale of today's data creation explosion is closely aligned to the distributed computational resources of the expanding universe of distributed embedded systems and devices. Multiple Service Operators (MSOs), such as cable providers, are an example of companies that drive both the rapid growth and evolution of large-scale computational systems, consumer and business data, as well as the deployment of an increasing number of increasingly-powerful embedded processors.

Our work is motivated precisely by the idea that the ubiquitous adoption of embedded devices

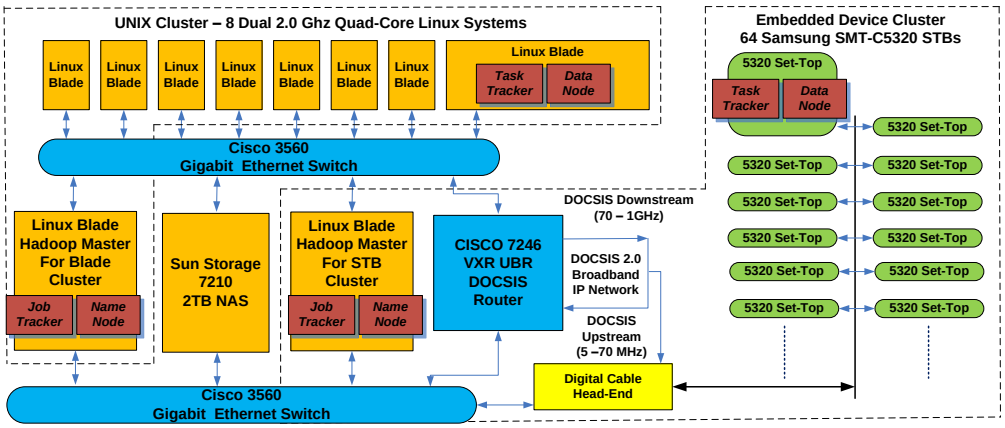


Figure 8: Architecture of the broadband embedded computing system for MapReduce utilizing Hadoop.

by consumers and the combination of the technology trends in embedded systems, data centers, and broadband networks open the way to a new class of heterogeneous Cloud computing for processing data-intensive applications. In particular, we propose a *broadband embedded computing system for MapReduce utilizing Hadoop* as an example of such systems. Its potential application domains include: ubiquitous social networking computing, large-scale data mining and analytics, and even some types of high-performance computing for scientific data analysis. We present a heterogeneous distributed system architecture which combines a traditional cluster of Linux blade servers with a cluster of embedded processors interconnected through a broadband network to offer massive MapReduce data-intensive processing potential (and, potentially, energy and cost efficiency).

3.2 The System Architecture

Fig. 8 provides an overview of the architecture of the system that we developed and built: this is a heterogeneous system that leverages a broadband network of embedded devices to execute MapReduce applications by utilizing Hadoop. It is composed of four main subsystems.

Linux Blade Cluster. The Linux Cluster consists of a traditional network of nine blade servers and a Network Attached Storage (NAS). Each blade has two quad-core 2GHz Xeon processors running Debian Linux with 32GB of memory and a 1Gb/s Ethernet interface. One of the nine blades is the Hadoop master host acting both as NameNode and JobTracker for the MapReduce runtime management [3]. Each of the other eight blades is a Hadoop slave node, acting both as DataNode and TaskTracker [3] while leveraging the combined computational power of the eight processing cores integrated on the blade. The blades use the Network File System (NFS) to mount the 2TB Sun storage array which provides a remote common file-system partition to store applications for each of the executing Hadoop MapReduce applications. For storing the Hadoop Distributed File System (HDFS) data, the blades use their own local hard-disk drive (HDD).

Embedded STB Cluster. The Embedded Cluster consists of 64 Samsung SMT-C5320 set-top boxes (STB) that are connected with a radiofrequency (RF) network for data delivery using MPEG and DOCSIS transport mechanisms. The Samsung SMT-C5320 is an advanced (2010-generation) STB featuring an SoC with a Broadcom MIPS 4000 class processor, a floating-point unit, dedicated video and 2D/3D-graphics processors with OpenGL support, 256MB of system memory,

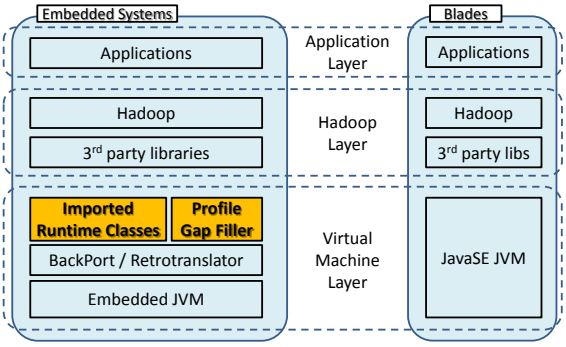


Figure 9: Two software stacks to support Hadoop: STB vs. Linux Blade.

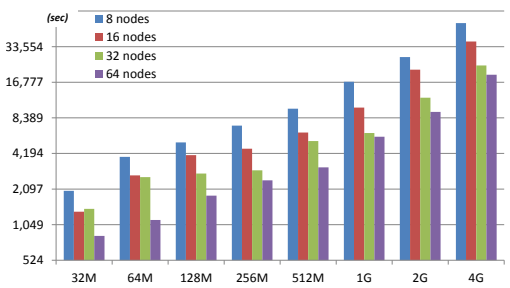
64MB internal Flash memory, 32GB of external Flash memory accessible through USB, and many network transport interfaces (DOCSIS 2.0, MPEG-2/4 and Ethernet). Indeed, an important architectural feature of modern STBs is the heterogeneous multi-core architecture design which allows the 400MHz MIPS processor, graphics/video processors, and network processors to operate in parallel over independent buses. Hence, user-interface applications (such as the electronic programming guides) can execute in parallel with any real-time video processing. From the viewpoint of running Hadoop applications as a slave node, however, each STB can leverage only the MIPS processor while acting both as DataNode and TaskTracker. [8] This is an important difference between the Embedded Cluster and the Linux Cluster. Finally, in each STB, a 32GB USB memory stick is used for HDFS data storage, while NFS is used for Java class storage.

Network. The system network is a managed dedicated broadband network which is divided into three IP subnets to isolate the traffic between the DOCSIS-based broadband Embedded Cluster network, the Linux Cluster network, and the digital cable head-end. Its implementation is based on two Cisco 3560 1Gb/s Ethernet switches and one Cisco 7246 DOCSIS broadband router. The upper switch in Fig. 8 interconnects the eight blades along with the NAS and master host. The lower switch aggregates all the components on the head-end subnetwork. The DOCSIS subnetwork is utilized by the Embedded Cluster whose traffic exists on both the Linux Cluster and the digital head-end network. The broadband router has 1Gb/s interfaces for interconnection to the Linux Cluster and head-end networks as well as a broadband interface for converting between the DOCSIS network and the Ethernet backbone. Each broadband router can support over 16,000 STBs, thus providing large-scale fan-out from the Linux Cluster to the Embedded Cluster.

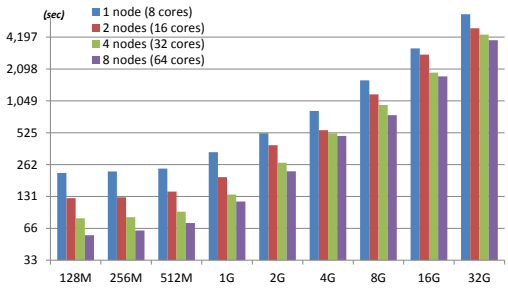
Embedded Middleware Stack. The embedded middleware stack is based on Tru2way, a standard platform deployed by major cable operators in U.S. as part of the Open Cable Application Platform (OCAP) developed in conjunction with Cablelabs [5]. Various services are delivered through the Tru2way platform including: chat, e-mails, electronic games, video on-demand (VOD), home shopping, interactive program guides, stock tickers, and, most importantly, web browsing [10]. To enable cable operators and other third-party developers to provide portable services, Tru2way includes middleware based on Java technology that is integrated into digital video recorders, STBs, TVs, and other media-related devices.

Tru2way is based on Java ME (Java Micro Edition) with CDC (Connected Device Configuration) designed for mobile and other embedded devices. The Tru2way standard follows FP

[8] In the Embedded Cluster, there is also a Linux blade which is the Hadoop master node, acting both as NameNode and JobTracker.



(a) Embedded Cluster (each node is one STB).



(b) Linux Cluster (each node is a 8-core blade.)

	# of Nodes STB / Blade	
Size	8 / 1	64 / 8
1G	49.2	49.4
2G	52.9	41.5
4G	63.9	39.6
8G	(64.5)	(48.8)
16G	(64.5)	(42.1)
32G	(61.3)	(38.3)

(c) Execution-time ratio.

Figure 10: *WordCount* execution time as function of problem size (bytes), node count: (a) Embedded Cluster and (b) Linux Cluster; (c) relative comparison.

(Foundation Profile) and PBP (Personal Basis Profile) including: io, lang, net, security, text, and util packages as well as awt, beans, and rmi packages, respectively. Additional packages include JavaTV for Xlet applications, JMF (Java Media Framework), which adds audio, video, and other time-based media functionalities, and MHP (Multimedia Home Platform), which comprises classes for interactive digital television applications. On top of these profiles, the OCAP API provides applications with Tru2way-specific classes related to hardware, media, and user-interface packages unique to cable-based broadband content-delivery systems.

Remark. While this rich set of Java profiles offer additional features to the embedded Java applications, there exists a significant gap between the Java stack provided by Tru2way and the Java Platform Standard Edition (Java SE), which is common to enterprise-class application development. Hence, since the standard Hadoop execution depends on the Java SE environment, we had to develop a new implementation of Hadoop specialized for the embedded software environment that characterizes devices such as STBs. We describe our effort in the next section.

3.3 Experiments

In order to evaluate our embedded Hadoop system for its scalability characteristics and execution performance, we executed a number of MapReduce experimental tests across the Linux Cluster and Embedded Cluster. All the experiments were performed while varying the degree of parallelism, i.e. by iteratively doubling the number of Hadoop nodes, of each cluster: specifically, from 1 to 8 Linux blades for the Linux Cluster (where each blade contains eight 2GHz processor cores) and from 8 through 64 STBs for the Embedded Cluster (where each STB contains one 400MHz processor core). The results can be organized in four groups which are presented in the following subsection. We report the average results after executing all tests multiple times.

The WordCount Application. *WordCount* is a typical MapReduce application that counts the occurrences of each word in a large collection of documents. The results reported in Fig. 10(a) and 10(b) show that this application scales consistently for both the Embedded Cluster and Linux Cluster. As the size of the input data increases, the Embedded Cluster clearly benefits from the availability of a larger number of STB nodes to process larger data sets. The Linux Cluster execution time remains approximately constant for data sizes growing from 128MB to 512MB since these are relatively small, but then it begins to double as the data sizes grow from 1GB to 32GB. In fact, above the 1GB threshold the amount of data that needs to be shuffled in the Reduce task begins to exceed the space available within the heap memory of each node. A similar transition from in-memory shuffling to in-disk shuffling occurs in the Embedded Cluster for smaller data sets due to the smaller memory available in the STB nodes: specifically, it occurs somewhere between 64MB and 512MB, depending on the particular number of nodes of each Embedded Cluster configuration.

Fig. 10(c) reports the ratios between the execution times of two Embedded Cluster configurations over two corresponding equivalent Linux Cluster configurations, for large input data sets. [9] The first column reports the ratio of the configuration with eight STBs over one single blade with eight processor cores; the second column reports the ratio of the Embedded Cluster configuration (with 64 STBs) over the Linux Cluster configuration (with eight blades for a total of 64 cores.) Across the different data sizes, the performance gap of the Embedded Cluster relative to the corresponding Linux Cluster with the same number of Hadoop nodes remain approximately constant: it is about 60 times slower for the configuration with 8 nodes and about 40 times slower for the one with 64 nodes. Notice that *these values are the actual measured execution times; they are not modified to account for the important differences among the two systems such as the 5X gap in the processor's clock frequency between the Linux blades and the STBs.*

HDFS & MapReduce Benchmarks. The second group of experiments involve the execution of a suite of standard Hadoop benchmarks. The goal is to compare how the performance of the Embedded Cluster and Linux Cluster scales for different MapReduce applications. The execution times of these applications expressed in seconds and measured for different configurations of the two clusters are reported in Table 4. The numbers next to the application names in the first column denote input parameters, which are specific to each application: e.g. "RandomTextWriter 8" denotes that the RandomTextWriter application is running eight mappers, while the "Pi-Estimator 1k" means that Pi-estimator runs with a 1k sample size.

Sleep is a program that simply keeps the processor in an idle state for one second, whenever a Map or a Reduce task should be executed. Hence, this allows us to estimate the performance overhead of running the Hadoop framework. For the representative case of running Sleep with 128 mappers and 16 reducers, the Embedded Cluster and the Linux Cluster performance is basically the same.

RandomTextWriter is an application that writes random text data to HDFS and, for instance, it can be configured to generate a total of 8GB of data uniformly distributed across all the Hadoop nodes. When it is running, eight mappers are launched on each Linux blade, i.e. one per processor core, while only one mapper is launched on each STB node. Since the I/O write operations dominate the execution time of this application, scaling up the number of processor cores while maintaining the size of the random text data constant does not really improve the overall execution

[9] The values in parenthesis are computed by extrapolating the execution times on the Embedded Cluster.

Benchmarks	8 STBs (8 cores)	64 STBs (64 cores)	1 Blades (8 cores)	8 Blades (64 cores)
Sleep	1285.1	119.6	1223.6	114.5
RandomTextWriter 8	799.6	743.9	177.6	172.0
PiEstimator 1k	461.1	163.5	212.1	52.5
PiEstimator 16k	463.4	474.0	213.7	52.5
PiEstimator 256k	603.6	783.2	214.6	52.4
PiEstimator 4M	1240.9	2048.2	213.9	52.5
PiEstimator 64M	7373.0	10482.5	314.8	58.4
K-Means 1G	3679.2	1149.3	794.7	245.0
Classification 1G	3009.0	784.9	864.7	254.5

Table 4: Execution times (in seconds) for various Hadoop benchmarks.

time.

Pi-Estimator is a MapReduce program that estimates the value of the π constant using the Monte-Carlo method [12]. For the Linux Cluster, the growth of the input size does not really impact the execution time for a given system configuration, while moving from a configuration with one blade to one with eight blades yields a 4x speedup. For the Embedded Cluster, in most cases scaling up the number of nodes causes higher execution times because this program requires that during the initialization phase the STBs receive a set of large class files which are not originally present in the Embedded Java Stack.

3.4 Summary

We developed, implemented, and tested a heterogeneous system to execute MapReduce applications by leveraging a broadband network of embedded STB devices. In doing so, we addressed various general challenges to successfully port the Hadoop framework to the embedded JVM environment. We completed a comprehensive set of experiments to evaluate our work by comparing various configurations of the prototype Embedded Cluster with a more traditional Linux Cluster. First, the results validate the feasibility of our idea as the Embedded Cluster successfully executes a variety of Hadoop applications. From a performance viewpoint, the Embedded Cluster typically trails the Linux Cluster, which can leverage more powerful resources in terms of processor, memory, I/O, and networking. On the other hand, for many applications both clusters demonstrate good performance scalability as we grow the number of Hadoop nodes. But a number of problems remain to be solved to raise the performance of executing MapReduce applications in the Embedded Cluster: in particular, critical areas of improvement include the STB I/O performance and the communication overhead among pairs of STBs in the DOCSIS broadband network. Still, the gap between embedded processors and blade processors in terms of speed, memory, and storage continues to decrease, while higher performance broadband networks are expected to integrate embedded devices into the Cloud. These technology trends hold the promise that future versions of our MapReduce computing system can help to leverage embedded devices for Internet-scale data mining and analysis.

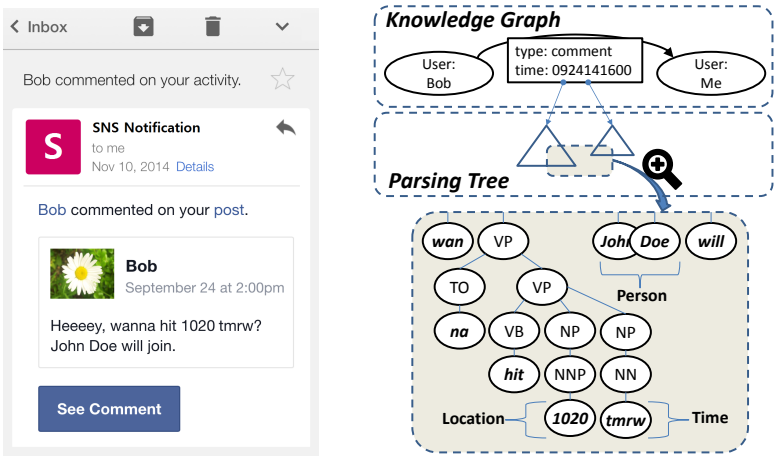
4 Algorithm-Division Reverse Offloading: Locally-Customized Training

Personal mobile devices offer a growing variety of personalized services that enrich considerably the user experience. This is made possible by increased access to personal information, which to a large extent is extracted from user email messages and archives. There are, however, two main issues. First, currently these services can be offered only by large web-service companies that can also deploy email services. Second, keeping a large amount of structured personal information on the cloud raises privacy concerns. To address these problems, together with Karl Stratos and Luca Carloni, I developed LN-Annote, a new method to extract personal information from the email that is locally available on mobile devices (without remote access to the cloud). LN-Annote enables third-party service providers to build a question-answering system on top of the local personal information without having to own the user data. In addition, LN-Annote mitigates the privacy concerns by keeping the structured personal information directly on the personal device. Our method is based on a named-entity recognizer trained in two separate steps: first using a common dataset on the cloud and then using a personal dataset in the mobile device at hand. Our contributions include also the optimization of the implementation of LN-Annote: in particular, we implemented an OpenCL version of the custom-training algorithm to leverage the Graphic Processing Unit (GPU) available on the mobile device. We present an extensive set of experiment results: besides proving the feasibility of our approach, they demonstrate its efficiency in terms of the named-entity extraction performance as well as the execution speed and the energy consumption spent in mobile devices.

4.1 Introduction

Recent advancements in personalized web-based services have enriched our daily lives. Intelligent personal assistant services such as Google Now [8] or Apple's Siri [4] can give "directions to home" or alert that "it's time to leave for your next meeting". Meanwhile, personal search services can answer queries based on the user's personal information. Googling "my flights", for instance, produces the upcoming flight reservations that the user has made. Personalized advertisement is another important (and most profitable) instance of personalized web services. The advertisement systems of Amazon, Facebook and Google [1, 6, 7] are known to utilize viewers' personal information such as previous purchase history.

What makes all these personalized services possible? The personal information collected by the service providers. Since its quality determines the quality of the personalized services, web service providers put in significant efforts to improve and extend its collection. One vast source of personal information is found in users' emails. Large web-service companies that provide also email services (like Google, Microsoft, and Yahoo) have the means to offer rich personalized services precisely thanks to the personal information they extract from the users' emails. The example of Fig. 11 illustrates this process. A notification email of a message posted on a Social Network Service (SNS) account by one of the user's friend is parsed through a sequence of steps to build structured data, including: 1) a knowledge graph indicating the subject, the object, a type of the action, and the contents of the comment; 2) the parsing tree of the comment; 3) various grammatical tags such as part-of-speech (e.g. Verbal Phrase and Noun Phrase) and Named-Entity



(a) An SNS Notification Email (b) Knowledge Graph and Parse Tree

Figure 11: Parsing email to collect personal information.

Recognition (NER) labels (e.g. Person, Location, and Time). This kind of structured personal information is stored and used later in various ways: e.g. to improve personal search services by retrieving results that are relevant to the named-entities related to the user.

Thanks to the growing amount of personal data that are available to be collected, it is easy to predict that personalized services will continue to evolve and expand. There are, however, limitations and concerns. First, the current methods of information extraction are not feasible for any small company that doesn't have its own email service because they are based on accessing large data sets collected with proprietary email services. Second, keeping large amount of structured personal information on centralized remote cloud servers raises privacy and security concerns [16, 77].

To address these problems, we present LN-Annote (Locally-customized NER-based Annotation), a novel information-extraction subsystem that is designed and optimized to process the email data available *locally* on each personal mobile device. Our contributions include:

- a distributed learning model based on two phases: *universal training* to generate a common parameter set on the cloud and *custom training* to refine and optimize the shared common parameter set by using the email data locally available on each mobile device;
- a discussion on how to extend the architecture of a personal search system to integrate the LN-Annote subsystem;
- an implementation of LN-Annote using locally available information and optimization methods leveraging the GPU on the mobile device; and
- an extensive set of experimental results to prove the feasibility, effectiveness, and efficiency of our approach.

4.2 LN-Annote System Design

In this section we present the design of LN-Annote, its main components, and how these interact with other modules as part of a bigger system. LN-Annote is an NER-based system optimized to extract information from email messages, in particular those sent via SNSs. The focus on

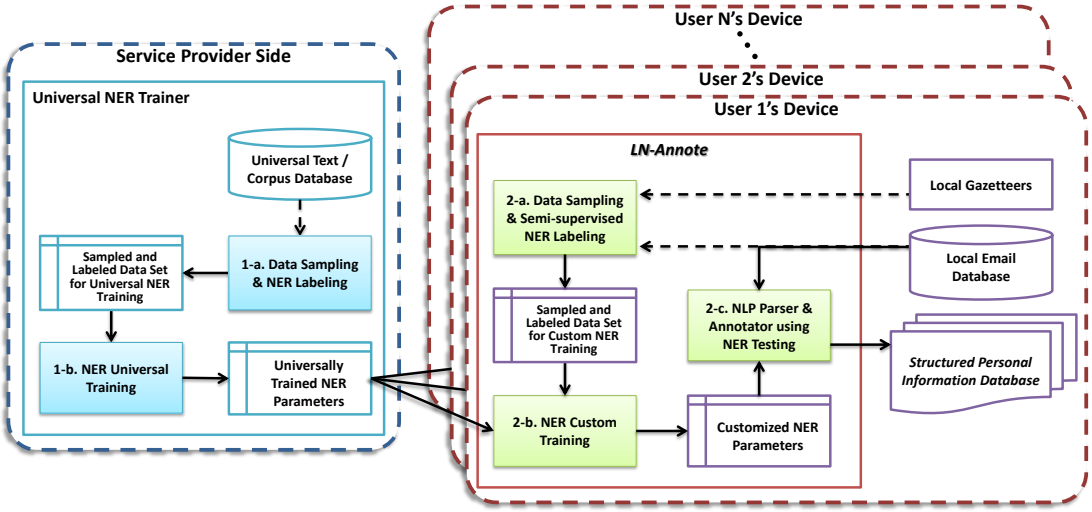


Figure 12: The flowchart of LN-Annote.

email is motivated by two observations: 1) email messages convey scads of personal information and 2) many web services send notification emails with very useful data such as reservations or recommendations.

Information Extraction and NER. Nowadays many companies send emails to their customers for various purposes such as discount offers, purchase history, appointment reminders, or activity updates on SNSs. As more companies integrate their services with email systems, these email messages contain a growing amount of personal information. Meanwhile, more and more people use their email to manage personal information [95]. Hence, the ability to extract personal information from emails becomes increasingly important. Nonetheless, existing extraction techniques have various limitations. One approach is to write vendor-specific parsing scripts; this, however, requires a large amount of manual labor to update the scripts whenever the vendor changes the email format. Another approach is to use Microdata embedded in the emails containing structured information [39]; this, however, is currently not very effective because the number of email messages that contain Microdata is very limited.

To overcome these problems, Natural Language Processing (NLP) techniques have been proposed to assist service providers in extracting useful information [71, 86]. Named-Entity Recognition (NER) is a popular NLP technique to classify given vocabularies into predefined categories [13, 70]. A wide variety of systems use NER for different text types such as queries, SNS posts, or résumés [38, 60, 72].

The performance of a NER system can be evaluated using various metrics. One of the most widely used metric is the F_1 *score* which is defined as the harmonic mean of *Precision* and *Recall*:

$$F_1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{1}$$

Precision, or *Positive Predictive Value*, is the correctness of the predicted classification and Recall, or *True Positive Rate*, is the coverage of the positive cases.

The LN-Annote System Workflow. To achieve more accurate prediction, we conceived a novel method that performs training for NER in two separate main steps, as illustrated in Fig. 12 (in this flowchart, a solid arrow represents a flow of data and a dashed arrow represents a sampling activity.) The first step, shown in the blue box (left), is *universal training*: this is essentially

identical to traditional learning and returns learning parameters that will be **shared among all users**. The second step, shown in the red boxes (right), is *custom training*: it runs on each personal device, takes the learning parameters from the universal training, and enhances them by further training with the **locally accessible dataset which is specific to each user**. The main goal of our method is to produce locally-customized learning parameters that work well for the particular local environment. However, the parameters need to perform well also on the global texts, by preserving the knowledge from universal training.

As shown later, LN-Annote achieves extraction performance comparable to training for a combination of the global dataset and the personal dataset, while requiring a significantly smaller amount of computation. Next, we provide more details on the two main steps.

1. Universal Training in the Cloud. This step consists of two substeps:

1-a. *Data sampling and NER labeling* is a preparation activity that takes samples from a universal text database and creates labels for the sampled data to feed the supervised training of Substep 1-b. Choosing a representative dataset with an appropriate amount is an important task for the quality of the training [69]. The sampled data can be labeled using different methods: a) manual labeling, b) manual labeling and running semi-supervised learning, and c) running unsupervised learning [28, 87]. For the experiments of this research, we used the CoNLL03 dataset provided with manually labeled NER tags [84], while semi-supervised learning is commonly used for large datasets.

1-b. *NER universal training* uses supervised learning to process the labeled data produced by Substep 1-a. In traditional machine learning, the learning parameters created by this kind of algorithm are directly used to test the prediction of NER labels for the actual dataset. In our approach, instead, these learning parameters are shared to the multiple mobile devices for custom training.

2. Custom Training & Testing on the Mobile Device. This step consists of three substeps:

2-a. *Data sampling & semi-supervised NER labeling* works similarly to Substep 1-a to produce labeled data for Substep 2-b. Here, the inputs are text samples selected from the local email database used by the email app running on the mobile device. Also, in this case the manual labeling cannot be applied because it is infeasible to ask the user of the personal device to do it. Instead, we obtained *local gazetteers*, a list of named-entities from a reliable source [65]. This substep is performed automatically by using a semi-supervised learning algorithm based on the labels from the gazetteers.

2-b. *NER Custom Training* updates the NER parameters by learning from the labeled dataset generated by the emails on the mobile device. The use of updated parameters is expected to keep the same performance as the use of universally trained parameters on the global dataset, while delivering better performance on the local emails.

2-c. *NLP Parser & Annotator using NER*, the final substep of LN-Annote processes the emails on the mobile device. This is done using NER based on the parameters created in the Substep 2-b. The outcome is stored in the Structured Personal Information Database where it can be used by any personal services running on the mobile device, as long as this is allowed by the user.

Notice that each substep of LN-Annote occurs with a different frequency: universal training (Substeps 1-a and 1-b) is done only once on the cloud servers; custom training (Substeps 2-a and 2-b), i.e. obtaining gazetteers, takes place periodically, e.g. once a week or a month; and finally, the actual information extraction (Substep 2-c) runs rather frequently, e.g. everyday.

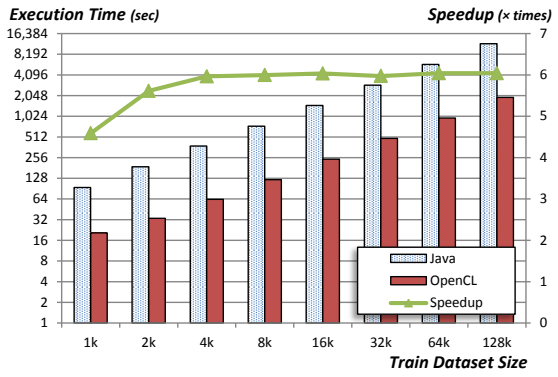


Figure 13: Execution time comparison (H=64).

4.3 Acceleration with Mobile GPUs

The execution of custom training on the personal mobile device poses some challenges in terms of increased energy consumption and the possible slowdown of the overall device. To address these challenges we performed two major optimizations.

4.4 Experiments

In the following experiments, we used the CoNLL03 shared task [84] in the universal training while we leveraged emails dataset in the custom training. CoNLL03 provides an English training dataset (*CoNLL03_train*) and two English test datasets (*CoNLL03_testa* and *CoNLL03_testb*). The actual email datasets used in the experiments were created with SNS notification emails, such as the example presented in Fig. 11(a), chosen from personally donated emails for research purposes. We created a training dataset and a testing dataset for each user and used these in Substeps 2-b and 2-c of Fig. 12, respectively. Let *email_traina* denote the training dataset from user$_a$ and *email_testa* denote the testing dataset from the same user. While a semi-supervised learning and gazetteers were used with the email datasets for training as explained in Section 4.2, the email datasets for testing were labeled following the CoNLL03 guidelines [84] for the experiments in this section. Each email dataset for custom training consists of 128,000 words out of which approximately 8,000 to 21,000 words are unique within that dataset. The used email dataset for testing contains around 64,000 words. Each measured value that has a possibility of variation, such as execution time and power consumption, was executed 10 times and averaged excluding the largest observation and the smallest observation.

OpenCL Performance Speedup. In this section, we evaluate our efforts on executing the NER training algorithm to relieve the increased CPU occupation and power consumption caused from having more computations on the mobile devices.

Fig. 13—15 compare the execution time of custom training by using the CPU and the GPU on the mobile device. The white bars represent the execution time of the CPU implementation written in Java. The red bars show the execution time of the GPU implementation written in OpenCL and embedded in the Java application through the Java Native Interface (JNI). The green curves indicate the speedup achieved by the OpenCL implementation (ran on the GPU) over the Java implementation (ran on the CPU).

Fig. 13 shows the execution time of the training algorithm when the hidden layer size H is set to

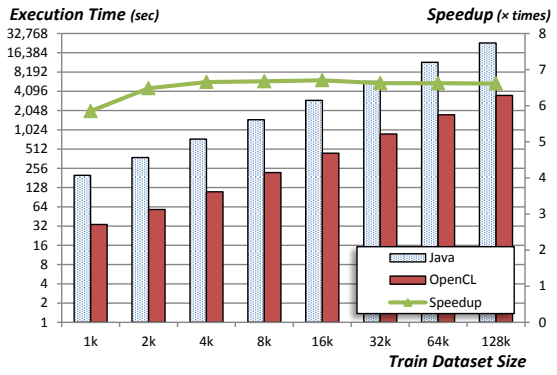


Figure 14: Execution time comparison (H=128).

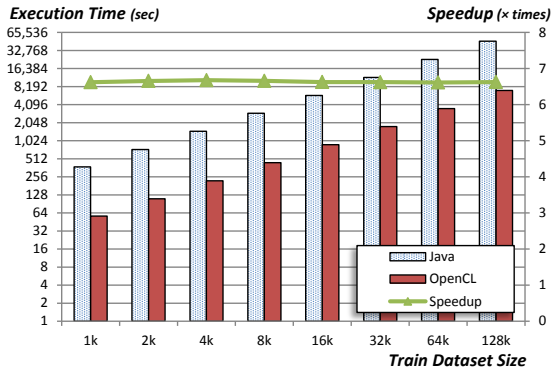


Figure 15: Execution time comparison (H=256).

64. The execution times of both Java and OpenCL implementations grow proportional to the input size, or the number of vocabularies in the training data set. In most cases, the speedup is close to 6, while the speedup is 4.5 and 5.5 for train dataset sized 1k and 2k, respectively. This interesting phenomenon occurs because for a small dataset the overhead from the OpenCL kernel invocation takes a large portion of the total execution time. This overhead includes times for initializing the OpenCL context, compiling the OpenCL kernel, and copying kernel arguments.

Fig. 14 is the execution time when H is set to 128. While increasing H improves prediction performance, it takes more time to complete the computations. Compared to Fig. 13 the execution times are almost doubled. This means that the size of the hidden layer impacts the amount of computations proportionally. The lower points on the left end of the green curve are observed also in this figure. The bending slops is more gradual than in Fig. 13. The overall speedup from the previous figure is increased because the OpenCL performance gets better as H increases. This is because our OpenCL implementation exploits the benefit of the concurrent hardware threads, which execute the matrix operations parallelly.

Fig. 15 presents the execution time when H is 256. Since the amount of computations required in each iteration has increased because of the larger H value, the lower speedups observed in the previous figures do not appear in these experiments. The overall speedup remains the same without a large improvement with respect to the previous experiment.

OpenCL Energy Saving. Our next set of experiments is about energy consumption. Since we have achieved a good speedup by utilizing the mobile GPU, we also expect to see some reduction in energy consumption. To measure the power consumption of each application and component, we used an open source software called *PowerTutor* [98] and a profiling tool, called *Trepn*, provided

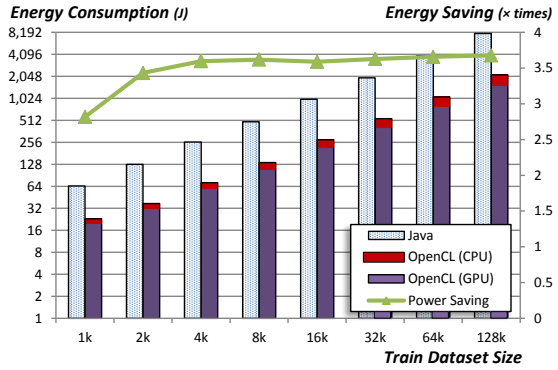


Figure 16: Energy consumption comparison (H=64).

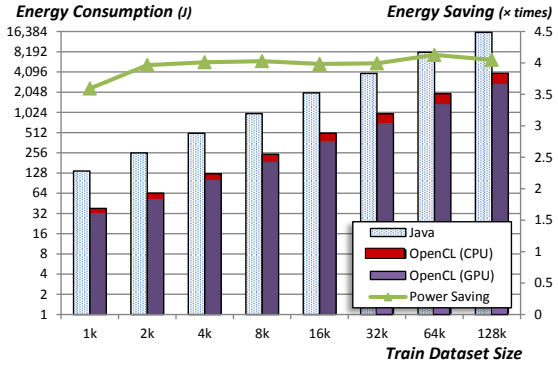


Figure 17: Energy consumption comparison (H=128).

by the chipset vendor [75].

Fig. 16 shows the energy consumed by the Java implementation executed on the CPU and by the OpenCL implementation run on the GPU. The red portion of the energy consumption of the OpenCL implementation is spent by the CPU while the rest is spent by the GPU. These experiments confirm that the energy consumed on the training for the same amount of input dataset could be reduced to one fourth by using the mobile GPU. This reduced energy consumption mostly came from the decreased execution time by the GPU. In fact, the power dissipation is even higher while the algorithm runs on the GPU. For instance, when the algorithm runs on the CPU the average power consumption by the CPU is approximately 0.682 Watt. On the other hand, the average power consumptions by the GPU and the CPU when the algorithm is executed on the GPU are around 1.02 Watt and 0.108 Watt, respectively. Another interesting point is that the power dissipation on the CPU, while the GPU is in use, is very low. This is because our OpenCL implementation is one consolidated kernel, thus not relying on the CPU during the entire iteration.[10] We speculate that the 0.108 Watt is mostly spent on the static power dissipation and on the maintenance of the Android app main thread, including event handling of the user interface.

Both Fig. 16 and Fig. 17 show lower energy savings for small datasets, i.e. 1k and 2k. Compared to the green curves in Fig. 13 and Fig. 14, the green curves in Fig. 16 and Fig. 17 stagger slightly. We presume that this is due to small errors introduced by the power measurement tools we used.

[10]There exist some GPUs that consume CPU resources even inside the GPU kernel, although the GPU we used in the experiments does not.

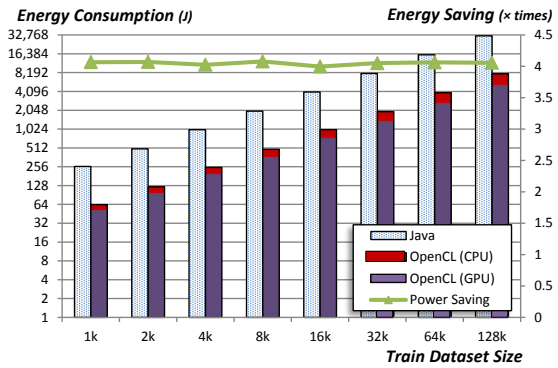


Figure 18: Energy consumption comparison (H=256).

4.5 Summary

We proposed and implemented Locally-customized NER-based Annotation (LN-Annote), a new method to extract personal information from emails stored locally on personal mobile devices. Our implementation is based on a newly-designed neural network model that works well for the two main phases of learning that characterize LN-Annote: universal training (performed in the cloud) and custom training & testing (performed on the mobile devices). The experimental results show the feasibility and effectiveness of LN-Annote. In particular, they demonstrate how the use of custom trained parameters actually improves the performance of NER on the local email data without reducing its performance on the dataset used for universal training.

5 Related Work

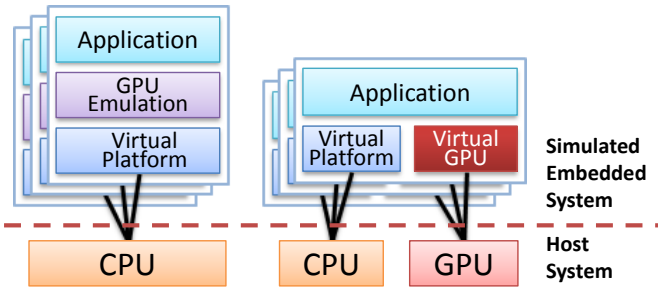
Design Tools. A number of studies have previously focused on helping system architects to better design distributed embedded systems by providing ways to optimize the process scheduling and the communication protocols [47, 74], tools to ease design space explorations [78, 51, 41], estimation models [96], network behavior simulations [32], and methodologies [55, 41].

Embedded Clusters. Our work is aligned with efforts in the Mobile Space to bridge MapReduce execution to embedded systems and devices. For example, the Misco system implements a novel framework for integrating smartphone devices for MapReduce computation [26]. Similarly, Elespuro *et al.* developed a system for executing MapReduce using smartphones under the coordination of a Web-based service framework [27].

Information Extraction Techniques. LN-Annote is related to some existing studies that focus on enhancing the model parameters for distinct cases. For instance, domain adaptation is an approach to transfer the feature vectors obtained on the source domain to the target domain [19]. In speech recognition, speaker adaptation is used to to improve the recognition performance by adapting the parameters of the acoustic models to better match the specific speaker's voice [76].

6 Research Plan

The following summarizes my research activities to date towards the completion of my dissertation: First, I built a design tool for large-scale, heterogeneous MCC systems. This tool eases the



(a) Simulation with Host CPU

(b) Simulation with Host CPU and GPU (Our Approach)

Figure 19: Two ways of simulating GPU applications.

development of MCC applications. Second, I developed reverse distributed offloading by building a computing framework for a cluster of embedded systems. In this system, the computational tasks from users are distributed over the cluster. Third, I developed an MCC application where algorithm-division reverse offloading plays a key role to reduce offloaded computations.

The following subsections provide more details on my research plan to complete the reminder of the dissertation. In Section 6.1, I explain how I plan to enhance the design tool for the applications that use mobile GPUs. In Section 6.2, I sketch how to develop query-division reverse offloading to optimize a new MCC application.

In Section 6.3, I present a tentative timeline for the completion of my research.

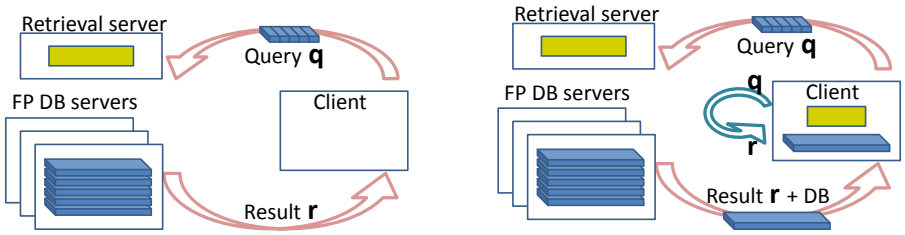
6.1 Improving the design tool for mobile GPU simulations

Despite their proliferation across many embedded platforms, GPUs present still many challenges to embedded-system designers. In particular, GPU-optimized software actually slows down the execution of embedded applications on system simulators. This problem is worse for concurrent simulations of multiple instances of embedded devices equipped with GPUs. To address this challenge, I plan to improve the design tool I developed by accelerating concurrent simulations on multiple virtual platforms. This technique leverages the physical GPUs present on the host machines, as shown in Fig. 19. The tool multiplexes the host GPUs to speed up the concurrent simulations without requiring any change to the original GPU-optimized application code.

Since analysis of execution times and energy consumption of the target application are critical features for modern system simulators, I plan to collect profile information from executing GPU code on the host GPUs and estimate the number of executed instructions on the target GPUs. These features will allow my design and simulation tool to support MCC applications that use mobile GPUs.

6.2 Query-Division Reverse Offloading: a ranking model for ASR

Audio Stream Retrieval (ASR) is an emerging class of applications in audio retrieval. ASR clients periodically query an audio database with an audio segment taken from the input audio stream to keep track of the original content sources in the stream or to compare two differently edited audio streams. We recently developed ASR applications such as broadcast monitoring systems, automatic caption fetching systems, and automatic media edit tracking systems. In these automated



(a) Non-optimized Querying (b) Optimized Querying

Figure 20: Querying optimization.

systems, ranking the retrieved result candidates is particularly important because the systems take the most likely result (top-ranked) for their purpose.

I propose query-division reverse offloading applied to a ranking model for ASR, as shown in Fig 20. Fig. 20(a) illustrates the iterative process of ASR without reverse offloading. The client queries the retrieval server. The server has various features such as query processing, retrieval from database, and ranking, collectively represented as a yellow box. This process can be improved by letting the client have a certain amount of fingerprints from a continuous content source and query the local database. In Fig. 20(b), the client has a 'light' version of search and rank functions (yellow box). First, the client still queries the server. When the retrieval server finds some results to answer the query, it sends the client a fingerprint sequence file along with the results. After a query interval, the client queries its own local search engine instead of querying the server. It can decide if the next result is still in the local database by comparing the score of the top-ranked result from the local search to a configurable threshold ϵ. If a good result is found within the local database, it simply returns this result. Otherwise, it forwards the query to the server and lets the server retrieve the queried fingerprint from the entire database. This approach significantly reduces the number of queries made to the servers.

6.3 Research plan

Table 5 shows my plan for completion of my PhD research. Thus, I plan to defend my thesis in October 2015.

Timeline	Work	Progress
2012	Development of an embedded cluster (Reverse Distributed Offloading)	completed
2013	Development of NETSHIP (a MCC CAD tool)	completed
2014	Development of LN-Annote (Algorithm-Division Reverse Offloading)	completed
Apr. 2015	Improvement of NETSHIP with mobile GPU simulation support	ongoing
Aug. 2015	Development of an ASR ranking model (Query-Division Reverse Offloading)	ongoing
Sep. 2015	Thesis writing	
Oct. 2015	Thesis defense	

Table 5: Plan for completion of my research

References

[1] Amazon Product Ads (www.amazon.com/productads).

[2] Android (developer.android.com).

[3] Apache hadoop (hadoop.apache.org).

[4] Apple siri (www.apple.com/ios/siri).

[5] Cablelabs (www.cablelabs.comi).

[6] Facebook Ads (www.facebook.com/ads).

[7] Google Adsense (www.google.com/adsense).

[8] Google Now (www.google.com/landing/now).

[9] Open Virtual Platforms (www.ovpworld.org).

[10] Tru2way (www.tru2way.com).

[11] S. Abolfazli et al. Cloud-based augmentation for mobile devices: Motivation, taxonomies, and open challenges. *IEEE Comm. Surveys Tutorials*, 16(1):337–368, Feb. 2014.

[12] J. Arndt and C. Haenel. *Pi-Unleashed.* 2001.

[13] H. Assal et al. Partnering enhanced-NLP with semantic analysis in support of information extraction. In *Proc. of the Int. Workshop on Ontology-Driven SW Engineering*, pages 9:1–9:7, Oct. 2010.

[14] F. Bellard. QEMU, a fast and portable dynamic translator. In *Proc. of the USENIX Annual Technical Conf.*, pages 41–46, Feb. 2005.

[15] T.-K. Chang. A secure cloud-based payment model for m-commerce. In *Proc. of the Int. Conf. on Parallel Proc.*, pages 1082–1086, Oct. 2013.

[16] D. Chaum. Security without identification: Transaction systems to make big brother obsolete. *Comm. of ACM*, 28(10):1030–1044, Oct. 1985.

[17] S. Chen, Y. Wang, and M. Pedram. Concurrent placement, capacity provisioning, and request flow control for a distributed cloud infrastructure. In *Proc. of the Conf. on Design, Automation & Test in Europe*, pages 279:1–279:6, Mar. 2014.

[18] S. Chen, Y. Wang, and M. Pedram. Optimal offloading control for a mobile device based on a realistic battery model and semi-Markov decision process. In *Proc. of the Int. Conf. on Computer-Aided Design*, pages 369–375, Nov. 2014.

[19] L. Chiticariu et al. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proc. of the Conf. on Empirical Methods in Natural Language Proc.*, pages 1002–1012, Oct. 2010.

[20] B.-G. Chun and P. Maniatis. Dynamically partitioning applications between weak devices and clouds. In *Proc. of the Workshop on Mobile Cloud Comp. & Services: Social Networks and Beyond*, pages 7:1–7:5, June 2010.

[21] A. Cidon et al. MARS: Adaptive remote execution for multi-threaded mobile devices. In *Proc. of the Workshop on Networking, Syst., and App. on Mobile Handhelds*, pages 1:1–1:6, Oct. 2011.

[22] C. Dall and J. Nieh. KVM for ARM. In *Proc. of the Linux Symp.*, pages 45–56, July 2010.

[23] M. D.Angelo et al. A simulator based on QEMU and SystemC for robustness testing of a networked Linux-based fire detection and alarm system. In *Proc. of the Conf. on* $ERTS^2$, pages 1–9, Feb. 2012.

[24] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Comm. of the ACM*, 51(1):107–113, Jan. 2008.

[25] Y. Ding et al. Nao: A framework to enable efficient mobile offloading. In *Proc. of the Workshop on Posters and Demos Track*, pages 8:1–8:2, Dec. 2011.

[26] A. Dou et al. Misco: a MapReduce framework for mobile systems. In *Proc. of the 3rd Int. Conf. on Pervasive Tech. Related to Assistive Environments*, pages 32:1–32:8, June 2010.

[27] P. R. Elespuru, S. Shakya, and S. Mishra. MapReduce system over heterogeneous mobile devices. In *Proc. of the Int. Workshop on SW Tech. for Embedded and Ubiquitous Syst.*, pages 168–179, Nov. 2009.

[28] D. Erhan et al. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, Mar. 2010.

[29] B. Gao et al. From mobiles to clouds: Developing energy-aware offloading strategies for workflows. In *Proc. of the Int. Conf. on Grid Comp.*, pages 139–146, Sept. 2012.

[30] Gartner. Forecast: Devices by operating system and user type, worldwide, Oct. 2013.

[31] Gartner. Forecast overview: Public cloud services, worldwide, Feb. 2013.

[32] E. Giordano et al. MoViT: the mobile network virtualized testbed. In *Proc. of the Int Workshop on Vehicular Inter-Net., Syst., and App.*, pages 3–12, June 2012.

[33] B. Girod et al. Mobile visual search. *IEEE Signal Proc. Magazine*, 28(4):61–76, July 2011.

[34] N. Govindarajan et al. Event processing across edge and the cloud for internet of things applications. In *Proc. of the Int. Conf. on Management of Data*, pages 101–104, Dec. 2014.

[35] R. L. Graham, T. S. Woodall, and J. M. Squyres. Open MPI: a flexible high performance MPI. In *Proc. of the Int. Conf. on Parallel Proc. and Applied Math.*, pages 228–239, Sept. 2006.

[36] A. Greenberg et al. The cost of a cloud: Research problems in data center networks. *SIGCOMM Comp. Comm. Review*, 39(1):68–73, Dec. 2008.

[37] A. Greenberg et al. Towards a next generation data center architecture: Scalability and commoditization. In *Proc. of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, pages 57–62, Aug. 2008.

[38] J. Guo et al. Named entity recognition in query. In *Proc. of the Int. Conf. on Research and Development in Info. Retrieval*, pages 267–274, July 2009.

[39] M. Heinrich and M. Gaedke. Data binding for standard-based web applications. In *Proc. of the Symp. on Applied Comp.*, pages 652–657, Mar. 2012.

[40] S. Howard and J. Martin. DOCSIS performance evaluation: piggybacking versus concatenation. In *Proc. of Southeast Regional Conf.*, volume 2, pages 43–48, Mar. 2005.

[41] Z.-M. Hsu, J.-C. Yeh, and I.-Y. Chuang. An accurate system architecture refinement methodology with mixed abstraction-level virtual platform. In *Proc. of the Design, Automation and Test in Europe Conf.*, pages 568–573, Mar. 2010.

[42] G. C. Hunt and M. L. Scott. The Coign automatic distributed partitioning system. In *Proc. of the Symp. on Operating Syst. Design and Impl.*, pages 187–200, Feb. 1999.

[43] IDC. Extracting value from chaos, June 2011.

[44] IDC. Smart connected device tracker, Mar. 2013.

[45] IDC. Worldwide datacenter census and construction 2014-2018 forecast: Aging enterprise datacenters and the accelerating service provider buildout, Oct. 2014.

[46] M. Ismail. WiMAX: a competing or complementary technology to 3G? In *Proc. of the Conf. on Integrated Circuits and Syst. Design*, pages 3–3, Sept. 2007.

[47] V. Izosimov et al. Design optimization of time-and cost-constrained fault-tolerant distributed embedded systems. In *Proc. of the Design, Automation and Test in Europe Conf.*, pages 864–869, Mar. 2005.

[48] Y. Jung, R. Neill, and L. P. Carloni. A broadband embedded computing system for MapReduce utilizing hadoop. In *Proc. of the Int. Conf. on Cloud Comp. Tech. and Science*, pages 1–9, Dec. 2012.

[49] Y. Jung, J. Park, M. Petracca, and L. P. Carloni. netShip: A networked virtual platform for large-scale heterogeneous distributed embedded systems. In *Proc. of the Design Automation Conf.*, pages 169:1–169:10, May 2013.

[50] Y. Jung, K. Stratos, and L. P. Carloni. LN-Annote: An alternative approach to information extraction from emails using locally-customized named-entity recognition. In *Proc. of the Int. World Wide Web Conf.*, May 2015. To appear.

[51] D.-I. Kang et al. A software synthesis tool for distributed embedded system design. In *Proc. of the Workshop on Languages, Compilers, and Tools for Embed. Syst.*, pages 87–95, May 1999.

[52] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Proc. of the Symp. on Discrete Algorithms*, pages 938–948, Jan. 2010.

[53] B. Kehoe et al. A survey of research on cloud robotics and automation. *Trans. on Automation Science and Engineering*, PP(99):1–12, Jan. 2015.

[54] D. Koutsonikolas and Y. C. Hu. On the feasibility of bandwidth estimation in wireless access networks. *Wireless Net.*, 17(6):1561–1580, Aug. 2011.

[55] J. Kruse et al. Introducing flexible quantity contracts into distributed SoC and embedded system design processes. In *Proc. of the Design, Automation and Test in Europe Conf.*, pages 938–943, Mar. 2005.

[56] A. Kumar and E. Pilli. University wide m-learning using cloud environment. In *Proc. of the Int. Symp. on Cloud and Services Comp.*, pages 118–123, Dec. 2012.

[57] D. R. Kumar. and S. ManjupPriya. Cloud based m-healthcare emergency using SPOC. In *Proc. of the Int. Conf. on Advanced Comp.*, pages 286–292, Dec. 2013.

[58] K. Kumar and Y.-H. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, Apr. 2010.

[59] S.-M. Lee et al. Fine-grained I/O access control of the mobile devices based on the xen architecture. In *Proc. of the Int. Conf. on Mobile Comp. and Net.*, pages 273–284, Sept. 2009.

[60] C. Li et al. TwiNER: Named entity recognition in targeted twitter stream. In *Proc. of the Int. Conf. on Research and Development in Info. Retrieval*, pages 721–730, Aug. 2012.

[61] X. Li and G. Autran. Implementing an mobile agent platform for m-commerce. In *Proc. of the Int. Computer SW and App. Conf.*, volume 2, pages 40–45, July 2009.

[62] A. K. Lisa Benenson, Lisa Goffredi. Data center efficiency assessment. Technical report, Aug. 2014.

[63] F. Liu et al. Gearing resource-poor mobile devices with powerful clouds: architectures, challenges, and applications. *IEEE Wireless Comm.*, 20(3):14–22, June 2013.

[64] A. Mohammad, K. Mohiuddin, M. Irfan, and M. Moizuddin. Cloud the mainstay: Growth of social networks in mobile environment. In *Proc. of the Int. Conf. on Cloud Ubiquitous Comp. Emerging Tech.*, pages 14–19, Nov. 2013.

[65] D. Nadeau, P. D. Turney, and S. Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In *Proc. of the Int. Conf. on Advances in Artificial Intelligence*, pages 266–277, Dec. 2006.

[66] J. Nagle. Congestion control in IP/TCP internetworks. *SIGCOMM Comp. Comm. Review*, 14(4):11–17, Oct. 1984.

[67] R. Neill et al. Embedded processor virtualization for broadband grid computing. In *Proc. of the Int. Conf. on Grid Comp.*, pages 145–156, Sept. 2011.

[68] R. Neill, A. Shabarshin, and L. P. Carloni. A heterogeneous parallel system running open MPI on a broadband network of embedded set-top devices. In *Proc. of the ACM Int. Conf. on Comp. Frontiers*, pages 187–196, May 2010.

[69] H. M. Nguyen et al. An alternative approach to avoid overfitting for surrogate models. In *Proc. of the Winter Sim. Conf.*, pages 2765–2776, Dec. 2011.

[70] S. Orlando, F. Pizzolon, and G. Tolomei. SEED: A framework for extracting social events from press news. In *Proc. of the Int. Conf. on World Wide Web Companion*, pages 1285–1294, May 2013.

[71] W. Paik et al. Applying natural language processing (NLP) based metadata extraction to automatically acquire user preferences. In *Proc. of the Int. Conf. on Knowledge Capture*, pages 116–122, Oct. 2001.

[72] S. Pawar, R. Srivastava, and G. K. Palshikar. Automatic gazette creation for named entity recognition and application to resume processing. In *Proc. of the Conf. on Intelligent & Scalable Syst. Tech.*, pages 15:1–15:7, Jan. 2012.

[73] R. Picek and M. Grcic. Evaluation of the potential use of m-learning in higher education. In *Proc. of the Int. Conf. on Info. Tech. Interfaces*, pages 63–68, June 2013.

[74] T. Pop, P. Eles, and Z. Peng. Design optimization of mixed time/event-triggered distributed embedded systems. In *Proc. of the Int. Conf. on HW/SW Codesign and Syst. Synthesis*, pages 83–89, Sept. 2003.

[75] Qualcomm Technologies Inc. Trepn: Profiler starter edition user guide. In *Qualcomm Developer Network*, pages 1–46. Apr. 2014.

[76] G. Saon et al. Speaker adaptation of neural network acoustic models using i-vectors. In *Proc. of the Workshop on Automatic Speech Recognition and Understanding*, pages 55–59, Dec. 2013.

[77] E. Sarigol, D. Garcia, and F. Schweitzer. Online privacy as a collective phenomenon. In *Proc. of the Conf. on Online Social Networks*, pages 95–106, Oct. 2014.

[78] D. E. Setliff, J. K. Strosnider, and J. A. Madriz. Towards a design assistant for distributed embedded systems. In *Proc. of the Int. Conf. on ASE*, pages 311–312, Nov. 1997.

[79] C. Shi et al. COSMOS: Computation offloading as a service for mobile devices. In *Proc. of the Int. Symp. on Mobile Ad Hoc Networking and Comp.*, pages 287–296, Aug. 2014.

[80] D. Taylor. Need for speed: PS3 Linux! *Linux Journal*, (156):5–6, Apr. 2007.

[81] The Guardian. Naked celebrity hack: security experts focus on iCloud backup theory, Sept. 2014.

[82] The Washington Post. 4chan: The 'shock post' site that hosted the private Jennifer Lawrence photos, Sept. 2014.

[83] The Washington Post. Proposed prince william data center prompts protest letter to Jeff Bezos, Jan. 2015.

[84] E. F. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proc. of the Conf. on Natural Lang. Learning*, pages 142–147, May 2003.

[85] S. S. Tsai, D. Chen, J. P. Singh, and B. Girod. Rate-efficient, real-time cd cover recognition on a camera-phone. In *Proc. of the ACM Intl. Conf. on Multimedia*, pages 1023–1024, Oct. 2008.

[86] J. Tsujii. Generic NLP technologies: Language, knowledge and information extraction. In *Proc. of the Annual Meeting on Assoc. for Comp. Linguistics*, pages 12–22, Oct. 2000.

[87] J. Turian, L. Ratinov, and Y. Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proc. of the Annual Meeting of the Assoc. for Comp. Linguistics*, pages 384–394, July 2010.

[88] C. H. K. van Berkel. Multi-core for mobile phones. In *Proc. of the Design, Automation and Test in Europe Conf.*, pages 1260–1265, Mar. 2009.

[89] C. Wang and Z. Li. Parametric analysis for adaptive computation offloading. *SIGPLAN Note*, 39(6):119–130, June 2004.

[90] S. Wang, Y. Liu, and S. Dey. Wireless network aware cloud scheduler for scalable cloud mobile gaming. In *Proc. of the Int. Conf. on Comm.*, pages 2081–2086, June 2012.

[91] X. Wang et al. SHIP: Scalable hierarchical power control for large-scale data centers. In *Proc. of the Int. Conf. on Parallel Arch. and Compilation Tech.*, pages 91–100, Sept. 2009.

[92] X. Wang et al. AppMobiCloud: Improving mobile web applications by mobile-cloud convergence. In *Proc. of the Asia-Pacific Symp. on Internetware*, pages 14:1–14:10, Oct. 2013.

[93] Y. Wang, X. Lin, and M. Pedram. A nested two stage game-based optimization framework in mobile cloud computing system. In *Proc. of the Int. Symp. on Service Oriented System Engineering*, pages 494–502, Mar. 2013.

[94] Y. Wang, J. Wu, and W.-S. Yang. Cloud-based multicasting with feedback in mobile social networks. *IEEE Trans. on Wireless Comm.*, 12(12):6043–6053, Dec. 2013.

[95] S. Whittaker, V. Bellotti, and J. Gwizdka. Email in personal information management. *Comm. of the ACM*, 49(1):68–73, Jan. 2006.

[96] Y. Xiangzhan et al. Research on performance estimation model of distributed network simulation based on PDNS conservative synchronization mechanism in complex environment. In *Proc. of the Int. Conf. on Comp. and Info. Tech.*, pages 2576–2580, Dec. 2010.

[97] K. Yang, S. Ou, and H.-H. Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *Comm. Magazine*, 46(1):56–63, Jan. 2008.

[98] L. Zhang et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proc. of the Int. Conf. on HW/SW Codesign and Syst. Synthesis*, pages 105–114, Oct. 2010.

[99] J. Zhou, X. Lin, X. Dong, and Z. Cao. PSMPA: Patient self-controllable and multi-level privacy-preserving cooperative authentication in distributed m-healthcare cloud computing system. *IEEE Trans. on Parallel and Dist. Syst.*, PP(99):1–11, 2014.