Cloud-Aided Design for Distributed Embedded Systems

YoungHoon Jung and Luca P. Carloni
Columbia University

Michele Petracca
Cadence Design Systems

Editor's notes:
This paper presents how to use cloud computing for designing distributed embedded systems. The cloud is used as a simulation platform. This platform allows the design and development of distributed embedded systems.

—Yung-Hsiang Lu, Purdue University

Cloud computing and embedded systems collaborate in the execution of many emerging classes of applications, while storing large amounts of data on the cloud. Examples of such applications include distributed-sensor data analysis, user behavior anticipation, and applications that run on smartphones. Typically, embedded systems act as widespread data collectors or user interface (UI) devices, while the cloud supports them with computation and storage services. Consequently, a growing amount of software involves computations that run concurrently on embedded devices and back-end clouds, which communicate through heterogeneous wireless and/or wired networks. (The "Related Work" section discusses various methods for the design-space exploration of distributed embedded systems.)

On the other hand, cloud computing can also contribute to the design of embedded systems. The increasing complexity of such systems requires engineers to run CAD tools that have heavy computation workloads. Cloud computing can help to more efficiently execute processes for simulation, optimization, and verification of embedded systems, from complex SoCs with billions of transistors to distributed embedded systems where heterogeneous networks connect various devices.

Motivated by these two aspects of the collaboration between embedded systems and cloud computing, we recently proposed the idea of a networked virtual platform (VP), which provides a simulation environment with high scalability and heterogeneity supports [1]. The simulation environment targets the design and testing of distributed embedded systems executing applications that can access cloud services. A networked VP can run on a cloud through the infrastructure as a service (IaaS) model. For the realization of our first prototype of a networked VP, NetShip, we defined the virtual-platform-on-virtual-machine (VP-on-VM) model. This model enables scalability along two key dimensions: horizontally, by adding more virtual machines (VMs), and vertically, by adding more VPs.

Thanks to these capabilities, NetShip can effectively support the design of large-scale software applications running on a heterogeneous network of multiple devices and cloud servers. In particular, it simplifies performance and scalability analysis by making it possible to simulate the execution of the actual applications and software stacks onto virtual

Digital Object Identifier 10.1109/MDAT.2014.2320521
Date of publication: 29 April 2014; date of current version: 22 July 2014.

Copublished by the IEEE CEDA, IEEE CASS, IEEE SSCS, and TTTC

models of the hardware and the network. Furthermore, the VP-on-VM model simplifies the deployment and migration of NetShip across the cloud because it supports the execution of a VM image on a cloud instance.

We have used NetShip in a case study to demonstrate how cloud-computing instances can be deployed to design and validate a complex distributed application that runs across portable devices and a cluster of servers. For this experiment, we could easily and rapidly deploy more than 100 VP instances, a task that would be unfeasible without the ability to leverage elastic cloud-computing services.

Networked virtual platforms

A VP is a simulation model of a system that provides virtual processors and peripherals and uses binary translation to execute the target binary code on top of a host instruction-set architecture (ISA). VPs enable system-level cosimulation of the hardware and software parts of a given system before the actual hardware implementation is finalized.

One VP instance is used to simulate and test a single device. Multiple VP instances can simulate multiple devices that form a physically connected system. In particular, a VP can be extended to support the model of peripherals and have network interface card (NIC) modules. Through a NIC module, a VP can communicate with other VPs in the network. A networked VP is realized precisely through the combination of multiple VP instances that run concurrently and interact with one another through their NIC modules. The resulting platform can serve as a full-system simulator of a distributed embedded system. It can run a real software stack to analyze the execution of various real applications before attempting an actual deployment of the physical machines.

Various challenges, however, must be addressed for the effective implementation of a networked VP. For example, the NIC module in each VP usually has no timing or performance model to precisely simulate the network communications. Also, the simulation time of each VP can progress independently of the others. Moreover, many distributed embedded systems today are large scale and heterogeneous, presenting further challenges. To address these challenges, we developed the VP-on-VM model.

A VM handles the management and provisioning of physical resources to create a virtualized environment. The resources are mostly provided by one or more server computers; the management is performed by a hypervisor. Examples of VPs include open virtual platform (OVP), virtual system platform (VSP), and quick emulator (QEMU). Examples of VMs include kernel-based virtual machine (KVM), VMware, and the instances enabled by the Xen hypervisor.

Our proposed VP-on-VM model supports the scalability of modeling and simulations. Multiple VP instances are hosted by the same VM, and multiple VM instances run in a networked VP. Figure 1 shows the example of a configuration where two VMs are running three VPs each. With the VP-on-VM model, the entire networked VP can be hosted on a set of VM instances provided by a private cloud, or by public cloud services such as Amazon's Elastic Compute Cloud (EC2) or Microsoft's Windows Azure.

Installing VP instances on VMs rather than on physical machines benefits from the various properties of cloud computing, such as easy management (duplication and deletion), migration, and monitoring of VM instance images. In particular, the

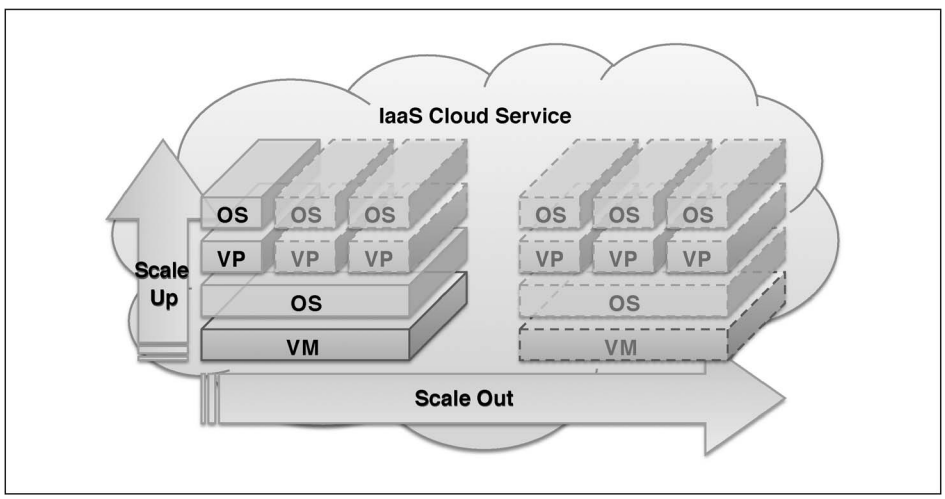

Figure 1. VP-on-VM model based on the IaaS model, and its scalability.

VP-on-VM model translates these properties into key advantages for the realization of a networked VP:

- the VM control panel simplifies the overall monitoring and management of physical resources;
- the networked VP's size can be quickly increased via preconfigured VM images;
- the automatic optimization of VP placement is possible through VM migration.

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.

Scalability

The VP-on-VM model makes the networked VP both horizontally and vertically scalable. As Figure 1 shows, users can scale the system out by adding more preconfigured VM instances to the network (horizontal scalability), and scale the system up by assigning more VPs to some of the running VM instances (vertical scalability). What makes vertical scalability possible is the VM dynamic configuration feature, which allows configuration changes such as adding more CPU cores and disk space. Meanwhile, horizontal scalability is obtained through the elastic VM creation feature in the IaaS model.

Heterogeneity

An important modeling aspect is support for heterogeneous system architectures. This comes in three different flavors. First, a system is heterogeneous when there are nodes with different types of processor cores, that is, based on different ISAs. For this type of heterogeneity, our framework lets different types of VPs be interconnected and interact through a network. This frees the networked VP from the limitation of each specific VP, while providing access to the superset of their features. For example, users interested in modeling an application running partly on certain ARM-based mobile phones and partly on MIPS-based servers can use this infrastructure to build a network of Android emulators (http://developer.android.com) and OVP nodes (http://www.ovpworld.org).

Second, a system with distinctly configured nodes is also heterogeneous. For example, a node equipped with additional GPUs is considered different from a node without a GPU, even if they have the same kind of CPU. The designer can vary node configurations using the specific configuration feature of each VP. Most VPs let designers configure the node with multiple different CPU cores and peripherals, including user-defined hardware accelerator modules.

Third, even two identically configured nodes differ when their interconnection network is different; for example, some nodes communicate via a particular wireless standard such as Global System for Mobile Communications (GSM) or Wi-Fi, whereas others communicate over Ethernet. These various network types have different network bandwidths, latencies, and error rates.

Leveraging the elasticity of the cloud

The encapsulation of multiple VPs in a VM instance lets us extend the various advantages of cloud computing to CAD of distributed embedded systems. The key is the ability to run many VM images as instances of elastic cloud computing. In cloud computing, elasticity denotes the ability to rapidly scale resources up and down on demand, an essential feature of public cloud platforms [2], [3]. With NetShip, a prebuilt VM image becomes a building block that can be deployed rapidly and efficiently on the cloud to support the simulation and analysis of a large-scale system. In particular, NetShip has two main advantages: scalability performance and maintenance.

Scaling out application services deployed on a cloud simply involves duplicating the VM image [4], [5]. Likewise, a VM image, configured for multiple VP simulations, can be programmatically duplicated as necessary. For example, a designer can scale out a simulation task from four to 32 embedded systems by invoking one command that handles the replication of the VM images and the launch of the VP instances in a few minutes. Without this automated scalability based on the cloud elasticity, the designer would have to manually perform many tedious and repetitive actions to install several physical machines, configure the operating systems, and copy the VP images. More generally, by relying on the cloud's elasticity, NetShip can be scaled horizontally and vertically to support the designer's needs. In particular, cloud computing is necessary to obtain vertical scalability, which is the ability to scale up a VM instance by allocating additional resources (for instance, dynamically adding a CPU core) and launching more VPs.

Cost of ownership and system maintenance are other issues that cloud computing greatly simplifies.

Otherwise, designers would have to physically possess all the machines necessary to run distributed simulations. For simulating very large-scale systems, this would be impractical due to the space availability, power supply costs, and heat management. Vendors of elastic-cloud-computing services take care of these issues, and cloud-aided design with tools such as NetShip lets designers rely on these services while focusing on the design itself.

Table 1 Time for adding new VPs using vCloud.

Results	No. of virtual platforms			
	1	4	16	64
Image size (Mbytes)	5,198	5,248	5,948	8,348
Time (s)	8	25	78	302

To build and run the distributed simulations for our experiments, we used a vSphere-based private cloud built with VMWare's vCloud Suite. However, users can easily port NetShip to any other type of vendor cloud system by modifying its VM management module. This module requests the cloud to duplicate, launch, terminate, or add resources such as CPU cores or disk space.

Table 1 shows the time required to add multiple VP instances using an API for cloud instance provisioning. Each VP uses a shared 2-GB read-only disk image and a dedicated 50-MB read-writable disk image. For example, adding 64 VPs to the simulation takes about 300 s to create a cloud instance that holds 64 VPs. The necessary work behind the cloud platform involves copying a thin-provisioned disk image with a size of 5148 MB + (64 × 50 MB) = 8348 MB and launching the newly created cloud instance. The size of the disk image for the cloud instance is 5148 MB, including 2048 MB of the shared image for the VPs. In our experiments, we use up to five VMs, each having four CPU cores with a 2.5-GHz clock frequency and a 4-GB main memory.

Prototyping a networked virtual platform

We developed NetShip as a prototype of a networked VP. In doing so, we also designed a general infrastructure for construction and management of networked VPs.

The main building blocks in NetShip are shown in Figure 2. NetShip can have various VP types. OVP is an industry-oriented platform for processor modeling, virtualization, and emulation that provides

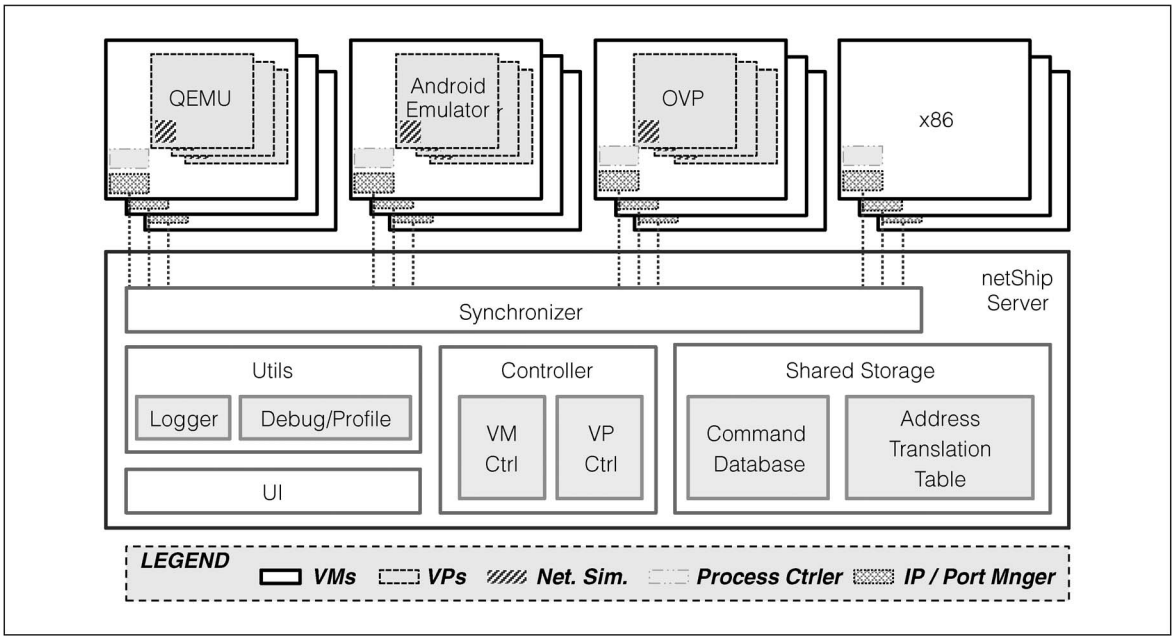

Figure 2. The architecture of NetShip.

open APIs. QEMU is an open-source processor emulator widely used for emulation and virtualization. Finally, the Android emulator is a QEMU-based mobile-device virtualization application that runs a full Android system stack. NetShip orchestrates the VP instances through the synchronizer.

Synchronizer

VPs vary in terms of the degree of accuracy of their timing models for the CPU performance that they support. Some VPs have no timing model and simply execute the binary code as fast as possible. This is often desirable, particularly when a VP runs in isolation. NetShip, however, runs multiple VPs on the same VM, so no VP can be allowed to monopolize CPU resources and starve other VPs. QEMU provides a crude way to keep simulation time within a few seconds of real time. 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, requiring 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 Figure 2 shows. The synchronizer is a single process that runs on just one particular VM and has a design that is similar to the fixed-time step synchronization method presented by D'Angelo et al. [6]. At each iteration, a central node increases the base timestamp, and the client nodes stop after reaching the given timestamp. However, because our target is distributed and scalable, we had to consider two additional aspects in our synchronizer:

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

NetShip targets large-scale systems involving software deployments across physically separated machines. The synchronization across these machines incurs milliseconds of delays during simulation. 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 the nanosecond level.

VM and VP management

Whereas the commands in the command database are dedicated to VP configuration, specialized modules manage the disk images of the VP and VM instances for creating, copying, and deleting. In particular, the VM controller shown in Figure 2 is integrated with the APIs provided by the cloud vendor that provisions and manages VM instances. The VM controller has a flexible design so that an implementation for additional cloud vendors can be plugged into the back-end layer. Our prototype has a default back-end implementation for VMWare's vCloud to automate the provision and control of VM instances from vCloud. Extending the VM controller module enables NetShip to be deployed on any type of cloud (for example, Amazon's EC2).

Network simulation

Some nodes can communicate via a particular wireless standard such as GSM or Wi-Fi, whereas others can communicate over Ethernet. The VP models of NetShip have their own NIC models. These NIC models, however, are purely behavioral and do not capture network performance [6]. Consequently, we developed a network simulation module (NetSim) 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 Figure 2 shows, NetSim resides in each particular VP and uses the traffic-shaping features based on the traffic control (tc) command, which manipulates the traffic control settings of the Linux kernel.

Cloud-Based crowd estimation system design

Here, we discuss a use case we conducted with NetShip. Crowd estimation, or crowd counting, is the problem of predicting how many people are present in a given area [7]. Several researchers have focused on crowd estimation based on the image processing of pictures [8], [9]. Using NetShip, we developed a crowd estimation application that processes pictures taken by mobile-phone users who are present in relatively wide areas (for example, a city or parts of a city).

We designed a distributed system that consists of embedded devices and a cloud-based server. Figure 3 illustrates the modeled system's design and implementation. In this scenario, smartphone users take some pictures (image source) with

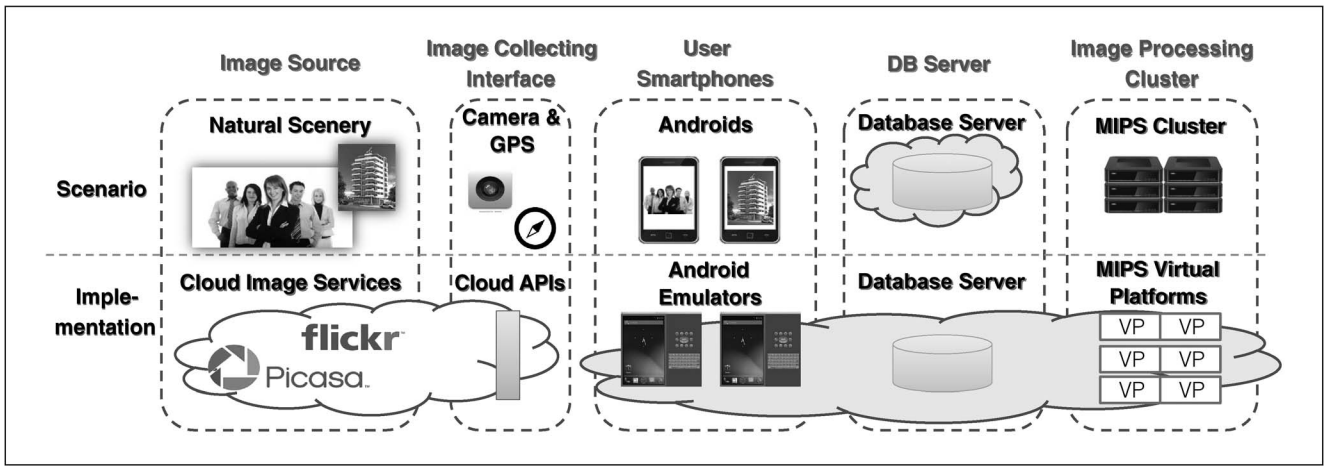

Figure 3. The design of the cloud-based crowd estimation system.

geolocation information using the camera module (image collecting interface) on the phone (user smartphones) and then upload these pictures to the database server in the cloud. A cluster of MIPS processors (image-processing cluster) downloads the pictures from a database server, runs an image-processing application to count the number of people in the picture, and adds the number to the geolocational sum in the database server.

Application design

The application iterates the following workflow:

1) the mobile-phone users take pictures and upload them to the image database, along with their geolocation;
2) the cluster of MIPS servers fetches one image at a time from the database and counts the people in this image using a human recognition algorithm;
3) the number of people in each image is stored back into the database;
4) the map generator creates a plotted image as the result.

Each iteration is done in parallel: the multiple Android emulators upload images, while the MIPS servers process the images.

In the implementation, several parts of the design are replaced by virtual counterparts. For instance, instead of physically deploying multiple Android phones, we use Android emulators running an application that simulates the behavior of smartphone users. Because of the lack of camera modules in the emulator, images downloaded from cloud image services (such as Picasa and Flickr) through their public cloud APIs serve as the user-taken pictures. Finally, OVP MIPS instances form a cluster to run an image-processing application.

The Android emulators and OVP instances in Figure 3 are VPs. Using NetShip, we built a networked VP that simulates the designed system.

Given the application requirements, we used NetShip to gain insights on the amount of resources required for real-time processing of the pictures taken by a large crowd in a particular geographic area: Manhattan, New York. Our main concern is the opportunity to build and study the networked VP and to use it to analyze the properties of the application that runs on it. In other words, we used this application primarily as a case study to test the capabilities of NetShip; the optimization of the crowd estimation quality was only a secondary concern.

Android emulator scalability

We used several Android emulators to model millions of mobile phones that sporadically take pictures (instead of using millions of emulators). To validate whether the emulators realistically reflect the actual devices' behavior with respect to network utilization, we performed multiple tests after making the following practical assumptions:

- there are three million mobile-phone users in Manhattan, and 2% of them upload two pictures per day;
- the uploading of pictures is evenly spread throughout the day (9:00 A.M. to 6:00 P.M.);
- the average image file size is 74 KB.

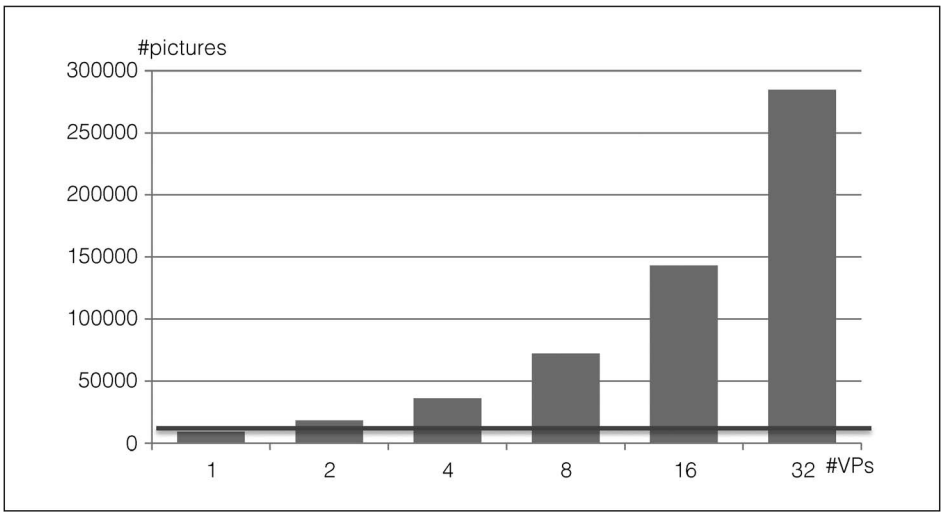

Figure 4. Maximum image-uploading capability as a function of the number of emulators.

On the basis of these assumptions, we estimated the number of pictures uploaded by the users in an hour as three million $\times$ 0.02 $\times$ 2/9 $\approx$ 13333. This value is represented by the bold lines in Figures 4 and 5. Thus, this is the number of pictures that the Android emulators must be able to upload and that the MIPS cluster must be able to process every hour. This requirement lets us dimension the system by deriving the minimum number of Android emulators and MIPS VPs that must be present in the cluster. For example, as Figure 4 shows, if the networked VP has only one Android emulator (first bar), it fails to upload 13333 pictures per hour because of the insufficient emulator performance. So, in this case, we must increase the number of emulators to at least two. Similarly, as Figure 5 shows, processing the uploaded pictures in real time requires increasing the cluster size to accommodate at least 32 MIPS VPs.

The incoming traffic measured from the database server is fairly constant, and is independent of the number of emulators that upload the same total number of pictures. This implies that we can analyze the system by using far fewer emulators than the number of smartphones we would have in reality (only four emulators versus three million smartphones), as long as those emulators can generate a comparable amount of traffic.

Bottleneck analysis

Given the average time required by one MIPS server to run the human recognition application for one picture, system designers can perform the following bottleneck analysis.

First, the designer can measure the number of MIPS servers required to support the volume of image processing for given input and output data rates. For example, assume the database server receives images from the cluster at a rate of S kb/s, and an MIPS server can execute the image-processing program for an average-sized image with a throughput of T kb/s. The designer can estimate that the system should have at least $\lceil S/T \rceil$ MIPS servers to guarantee real-time execution of the application.

Second, in certain circumstances it might not be possible to increase the number of available servers N or the average throughput T of each server. Consequently, the system can process the input data only at a

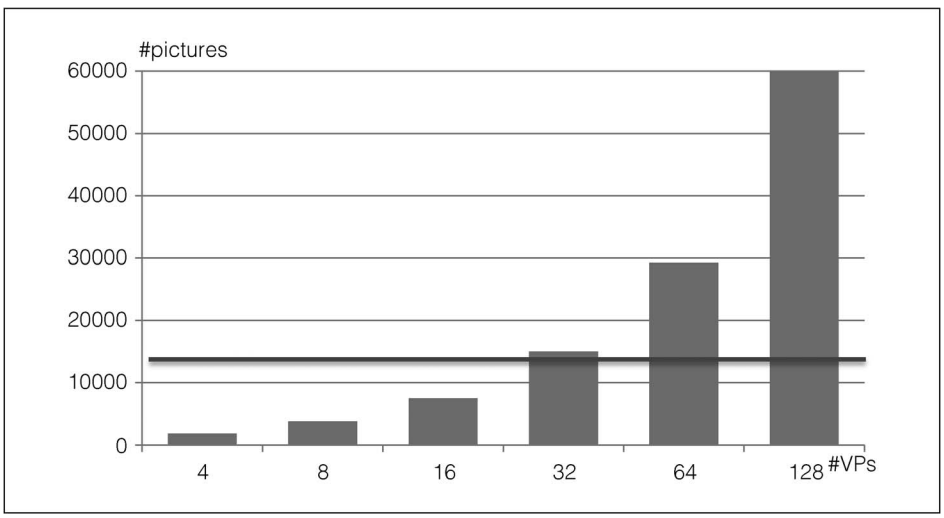

Figure 5. Image-processing capability as a function of the number of VPs.

rate S', smaller than the rate S at which images are received from the database server. In such cases, the designer can acquire precise indications from the simulation analysis to determine a new, sustainable image size for the application. Specifically, if the images arriving at the rate of S have an average size equal to I, then reducing this size down to $I' = I \times S'/S$ would make the application work in real time when computation resources cannot be increased.

Third, the network traffic through the database server includes picture uploading from mobile phones, picture downloading by the MIPS clusters, updating and reading of geolocation information, and the counting of people in images. On the basis of the network traffic analysis and the observation of how the behavior scales as the system grows, the designer can evaluate the best database architecture (for example, distributed versus centralized).

Related work

Several researchers have provided methods for the design-space exploration of distributed embedded systems [10], [11]. Some projects have focused on the simulation of specific classes of distributed embedded systems. For instance, the design of wireless sensor networks (WSNs) has benefited from the development of tools that provide better scalability [12], [13], accuracy [14], codesign of hardware and software [15], [16], and testbed provisioning [17].

Recent years have also seen the development of simulation frameworks for machine-to-machine (M2M) and distributed embedded systems, which leverage VMs [18], [19] and, by extension, cloud computing [20], [21].

In contrast to these earlier research projects, our approach leverages the properties of cloud computing to achieve unprecedented degrees of scalability and heterogeneity. In terms of scalability, our networked VP, NetShip, can simulate thousands of embedded devices that execute an actual complete software stack. In terms of heterogeneity, NetShip lets us model the different properties of heterogeneous networks in addition to those of heterogeneous CPUs and peripherals.

Networked virtual platforms can be used for various purposes, including simulation of distributed applications; systems, power, and performance analysis; and costs modeling and analysis of embedded networks' characteristics. In particular, since energy efficiency is becoming the most important concern for many classes of embedded applications, we plan to integrate models for power consumption into NetShip. Another important avenue of future research involves how to leverage the progressive enhancement of cloud computing in both quality and quantity in order to improve the power and time-modeling accuracy of networked VPs. ■

Acknowledgment

This work was supported in part by the National Science Foundation under Awards #644202 and #1147406, and by an Office of Naval Research (ONR) Young Investigator Award. The authors would like to thank Y. Watanabe for useful discussions.

■ References

[1] Y. Jung, J. Park, M. Petracca, and L. P. Carloni, "NetShip: A networked virtual platform for large-scale heterogeneous distributed embedded systems," in *Proc. Design Autom. Conf.*, 2013, article 169.

[2] P. C. Brebner, "Is your cloud elastic enough? Performance modelling the elasticity of infrastructure as a service (IaaS) cloud applications," in *Proc. 3rd ACM/SPEC Int. Conf. Performance Eng.*, 2012, pp. 263–266.

[3] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Elastic management of cluster-based services in the cloud," in *Proc. 1st Workshop Autom. Control Datacenters Clouds* 2009, pp. 19–24.

[4] vCloud API Programming Guide. [Online]. Available: http://pubs.vmware.com/vcd-51/index.jsp

[5] Amazon Elastic Compute Cloud Documentation. [Online]. Available: http://aws.amazon.com/documentation/ec2

[6] M. D'Angelo, A. Ferrari, O. Ogaard, C. Pinello, and A. Ulisse, "A simulator based on QEMU and SystemC for robustness testing of a networked linux-based fire detection and alarm system," in *Proc. Embedded Real Time Softw. Syst. Conf., Softw. Platform Integr. Eng. Things*, 2012. [Online]. Available: http://www.erts2012.org/Site/0P2RUC89/4B-3.pdf

[7] A. N. Marana, L. da Fontoura Costa, R. A. Lotufo, and S. A. Velastin, "Estimating crowd density with Minkowski fractal dimension," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 1999, vol. 6, pp. 3521–3524.

[8] T. Fei, L. SunDong, and G. Sen, ''A novel method of crowd estimation in public locations,'' in *Proc. Int. Conf. Future BioMed. Inf. Eng.*, 2009, pp. 339–342.

[9] W. Li, X. Wu, K. Matsumoto, and H.-A. Zhao, ''Crowd density estimation: An improved approach,'' in *Proc. IEEE 10th Int. Conf. Signal Process.*, 2010, pp. 1213–1216.

[10] D. E. Setliff, J. K. Strosnider, and J. A. Madriz, ''Towards a design assistant for distributed embedded systems,'' in *Proc. 12th IEEE Int. Conf. Autom. Softw. Eng.*, 1997, pp. 311–312.

[11] Z.-M. Hsu, J.-C. Yeh, and I.-Y. Chuang, ''An accurate system architecture refinement methodology with mixed abstraction-level virtual platform,'' in *Proc. Design Autom. Test Eur. Conf.*, 2010, pp. 568–573.

[12] P. Levis, N. Lee, M. Welsh, and D. Culler, ''TOSSIM: Accurate and scalable simulation of entire TinyOS applications,'' in *Proc. 1st Int. Conf. Embedded Netw. Sensor Syst.*, 2003, pp. 126–137.

[13] B. L. Titzer, D. K. Lee, and J. Palsberg, ''Avrora: Scalable sensor network simulation with precise timing,'' in *Proc. 4th Int. Symp. Inf. Process. Sensor Netw.*, 2005, pp. 477–482.

[14] L. Girod et al. ''Emstar: A software environment for developing and deploying heterogeneous sensor-actuator networks,'' *ACM Trans. Sensor Netw.*, vol. 3, no. 3, 2007, DOI: 10.1145/1267060.1267061.

[15] J. Zhang et al. ''A software-hardware emulator for sensor networks,'' in *Proc. 8th Annu. IEEE Commun. Soc. Conf. Sensor Mesh Ad Hoc Commun. Netw.*, 2011, pp. 440–448.

[16] S.-H. Lo et al. ''SEMU: A framework of simulation environment for wireless sensor networks with co-simulation model,'' in *Proc. 2nd Int. Conf. Adv. Grid Perv. Comput.*, 2007, pp. 672–677.

[17] G. Coulson et al. ''Flexible experimentation in wireless sensor networks,'' *Commun. ACM*, vol. 55, no. 1, pp. 82–90, 2012.

[18] P. Boyko and A. Mazo, ''Qemunet: An approach to an automated virtualized testbed,'' in *Proc. 4th Int. ICST Conf. Simul. Tools Tech.*, 2011, pp. 431–438.

[19] S.-H. Hung, C.-H. Chen, and C.-H. Tu, ''Performance evaluation of machine-to-machine (M2M) systems with virtual machines,'' in *Proc. 15th Int. Symp. Wireless Pers. Multimedia Commun.*, 2012, pp. 159–163.

[20] M. D. Rossetti and Y. Chen, ''A cloud computing architecture for supply chain network simulation,'' in *Proc. Winter Simul. Conf.*, 2012, article 284.

[21] K. Vanmechelen, S. De Munck, and J. Broeckhove, ''Conservative distributed discrete event simulation on amazon EC2,'' in *Proc. 12th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2012, pp. 853–860.

YoungHoon Jung is currently working toward a PhD in computer science at Columbia University, New York, NY, USA. His research focuses on distributed embedded systems and cloud systems, including applications, system optimization, and system simulation. Jung has an MBA in management information systems from Chungnam National University, Daejeon, Korea. He is a student member of the IEEE and the Association for Computing Machinery (ACM).

Michele Petracca is a member of the consulting staff at Cadence Design Systems, San Jose, CA, USA, where he works on technology and methodology development for the top-down design of electronic systems. His research focuses on design methodologies for complex SoCs. Petracca has a PhD in electronic engineering from Politecnico di Torino, Turin, Italy.

Luca P. Carloni is an Associate Professor in the Department of Computer Science, Columbia University, New York, NY, USA. His research interests include SoC platforms, multicore architectures, system-level design, embedded software, and distributed embedded systems. Carloni has a PhD in electrical engineering and computer sciences from the University of California Berkeley, Berkeley, CA, USA. He is a senior member of the IEEE and the Association for Computing Machinery (ACM).

■ Direct questions and comments about this article to YoungHoon Jung, Department of Computer Science, Columbia University, New York, NY 10027 USA; jung@cs.columbia.edu.