STRATUS: Assembling Virtual Platforms from Device Clouds

Minsung Jang, Karsten Schwan
Center for Experimental Research in Computer Systems
Georgia Institute of Technology
minsung@gatech.edu karsten.schwan@cc.gatech.edu

***Abstract*—There is an increasing number of network-enabled computing devices in homes and offices, driven by continued improvements in device capabilities and network connectivity. By exploiting the virtualization technologies that have begun to pervade even the mobile domain, these devices' hardware components, such as displays, input devices, disks, or processors, can be decoupled from the physical platforms on which they reside to form a *resource pool* or *device cloud*. By drawing on the composite resources of device clouds, applications can leverage the heterogeneity present in the cloud to exploit hardware/device differences in terms of power consumption, computational speeds, display sizes, or the presence of certain accelerators, and can take advantage of software diversity in terms of the different operating environments and applications that efficiently operate on individual devices. This paper implements and evaluates the concept of device clouds, in which virtual execution platforms dynamically composed from sets of devices are built for applications, using automated methods that are based on simple policies. Experimental results identify the basic overheads associated with device clouds and their use, and demonstrate the advantages of dynamically constructed virtual platforms rather than individual machines, both in terms of improvements in system properties like power usage and improvements in user experiences for media delivery and playout.**

Keywords-device cloud; virtualization; management; virtual platform; virtualized resources

I. INTRODUCTION

Today's mobile and home computing devices are highly diverse in terms of their hardware configurations and capabilities, ranging from media players like iPods, to mobile phones, large-screen TVs with set top boxes, notebooks or laptops, home PCs with convenient keyboards and large disks. While these network-enabled devices can interoperate with each other as well as with remote servers, current systems and software remain lacking in terms of their ability to freely combine their capabilities to form the virtual platforms that provide the most current value to end users. It should be straightforward, for instance, to take a movie resident on a mobile device and display it on a home's large-screen TV, as soon as the person using the device enters her home. It should be similarly easy to evict battery-consuming actions using a device on the mobile phone and instead, move them to a device attached to the home PC.

This paper describes the concept of virtual platforms, along with the Stratus framework that dynamically constructs - synthesizes - them from the sets of network-enabled devices, *device clouds*, present in end user environments like homes or offices. Driven by the management software and user policies shown in Fig. 1, with Stratus, virtual platforms are constructed in ways that:

- exploit the difference properties and capabilities of distributed devices to efficiently provide desired functionality, e.g., to minimize the use of battery energy or to improve performance by using the high definition(HD) video decoders/encoders already present in mobile systems along with the powerful processors and large disks in home PCs;
- provide improved experiences to end users, e.g., by substituting a high resolution for a low resolution display and/or by removing the need for user involvement to make good use of available devices;
- offload from end users the need to deal with manual composition actions such as device docking or complex steps for being slaving devices or similar actions, by providing a single device cloud within which they can utilize the aggregate resources of the many devices they routinely use in their homes or office environments[1][2].

Inherent properties of device clouds are: *(i)* they are comprised of sets of devices that can participate[2] in common and dynamically synthesized virtual platforms so as to deliver *(ii)* a seamless user experience across multiple devices and machines[3][4]; *(iii)* participating entities need not be entire machine, but may provide only some of their components, like displays or storage devices, toward the common virtual platform (i.e., device clouds); and *(iv)* virtual platforms are automatically constructed and continuously managed, to obtain the desired global properties stated by user policies, like minimal battery usage and/or the delivery of end user experiences at desired levels of quality. To illustrate, consider a user who wants to edit video clips using devices available in his home. Toward this end, the Stratus framework can construct a virtual platform optimized for clip handling, composed of say, the processor of the high end media laptop that has an MPEG decoder/encoder, the large disk on the home PC, and the TV as a display screen.

978-0-7695-4460-1/11 $26.00 © 2011 IEEE
DOI 10.1109/CLOUD.2011.70

Stratus shares with current cloud solutions its vendor-neutral approach of exploiting virtualization technology, by synthesizing and managing virtual platforms at hypervisor level.

Further, platforms are elastic in terms of their capabilities and performance, where elasticity is obtained by modifying virtual to physical component mappings and existing virtual machine migration. Elasticity is driven by end user desires expressed in user policies that determine the features a virtual platform should include (e.g., the policy for processing HD video clips is designed to establish a virtual platform that includes a HD accelerator and a large screen) and additional performance or power attributes (e.g., the platform should use minimal amounts of battery consumption).

Stratus uses policies along with information about current device availabilities to construct suitable platforms and in addition, to dynamically configure them when devices arrive or leave or when requirements change. The policies differ from prior work on SLA- or SLO-driven system management[10] in that they are used both for constructing and for managing platforms. A long battery life policy, for instance, might build platforms that minimize battery consumption, whereas a performance policy would create the fastest platforms possible. We use several such policies in the technical evaluations in Section III. Policies may be formulated by users or administrators, to enforce desired global properties or to act on behalf of certain classes of applications or for certain workload characteristics [7][8][9].

The current implementation of the device cloud extends the Xen[5] hypervisor present on all participating machines with additional control and data channels for device interaction and management. A control channel is used for commands and for status reports between participating Stratus nodes and the Stratus master. The master maintains the resource information about each participating node, evaluates resource capabilities, and constructs and configures virtual platforms built from these resources.

Performance evaluations in Section III demonstrate the utility of policy-driven platform construction and configuration, where simple policy changes can dramatically change virtual platforms into alternative configurations more suitable for those policies. Further, by exploiting device heterogeneity, performance can be improved beyond that of single machines or devices. Experimental results for instance, show a 92% reduction of time, a 11.7% reduction of power consumption, and a 15% battery life enhancement when operating on a Stratus virtual platform vs. on single devices.

The remainder of this paper is organized as follows. Section II presents an overview of a representative virtual platform generated by Stratus, to illustrate the utility of device clouds and virtual platforms and to explain platform construction and configuration. Section III experimentally demonstrates the relative advantages of distributed virtual to single physical platforms. Related work appears in Section IV, followed by conclusions and future work.

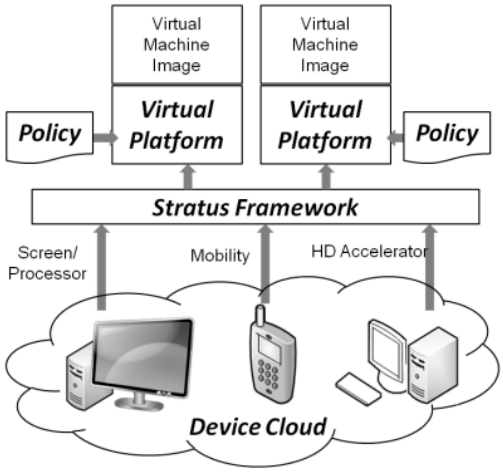

Figure 1. Stratus and Device Cloud

II. Device Clouds As Basis for Virtual Platforms

A. Motivation and Use Case

With increased heterogeneity in systems and devices accessible to end users, it becomes difficult to decide which devices may be most suitable for which tasks (e.g., because they may have 3D accelerators, gyro sensors, or GPS receivers), and then, to deal with device peculiarities and differences to carry out those tasks. The device cloud addresses these problems in two ways: *(i)* device composition into virtual platforms is independent of operating systems, middleware, or applications, because it is carried out at hypervisor level, and *(ii)* there is limited end user involvement, in that runtime composition and configuration are drive by user-defined policies that guide hypervisor's actions.

To illustrate, consider a user watching a video on a smartphone on the way home. Upon arrival, Stratus components detect additional devices available in the home. Guided by a user-specified policy that demands increased video quality, for instance, Stratus assembles a more appropriate virtual platform from the home's device cloud including home PCs with large monitors and disks (in which the desired video may already be present), in addition, the smartphone's input pad. To provide continuous service, the new virtual platform is activated by first migrating the virtual machine on the smartphone - containing all relevant playout state - to the home PC, then *re-wiring* the media store to the home PC's disk rather than streaming it via the mobile phone's radio. Alternatively, the same video could be fetched from a cloud-enabled media provider offering services to home users via say, their set top boxes, or from Internet TVs. Source selection is driven by user preferences stated in policies. An additional set of actions may even enable live editing of selected video frames present on the video, on a virtual platform composed on a PC or laptop also present in the home.

To offer the flexibility in device use and seamlessness of operation required to easily realize usage examples like those above, Stratus implements cooperative behaviors across the multiple per-platform hypervisors present on home devices, such as those on PCs, smartphones, or laptops. The purpose of cooperation is to better use all resources present in Stratus-enabled systems to provide to end users exactly the virtual platforms they desire. We use the term *device cloud* to emphasize the fact that any device attached to a Stratus-enabled machine can participate in a virtual platform, as long as the device itself (e.g., the wide-screen TV) is dynamically discoverable and shareable (e.g., because it is attached to a machine that can manage it via its hypervisor - such as the home PC or some media box). Stated differently, device cloud participants are those entities that are Stratus-enabled, which in our current implementation, requires them to be attached to or on machines that run hypervisors with embedded Stratus node support. One such node will act as Stratus master. One method to attain seamlessness in operation is via runtime virtual machine migration. This makes it possible to move arbitrary applications and systems, without having to depend on the availability of specific features of say, media players or editing software that permit internal state to be externalized. Ease of use is derived from the aforementioned user policies.

We next explain how we construct, maintain, and configure virtual platforms, and the basis on which they run, resulting in a user-accessible cloud of devices discovered by and participating in virtual platforms.

B. *Design Principles* and *Assumptions*

The hypervisor-level functionality of Stratus includes *(i)* Discovery: methods and mechanisms for runtime device discovery, inclusion, and exclusion in virtual platforms, *(ii)* Management: software for dynamic platform construction and configuration, including to monitor the current properties and capabilities of participating devices and thus, the composite properties of the virtual platforms into which they are assembled, and *(iii)* Automation: executable encodings of user policies to drive *(i)* and *(ii)*.

Additional useful functionality like authorization and rights management may show benefit in controlling and enforcing security policies for shared devices that may not be wholly owned by a certain user, the secure and reliable inclusion of open facilities like those offered by public clouds, and multi-site device clouds like those spanning multiple homes or offices. The current implementation addresses single, private device clouds, assumes that all participating devices to be trustworthy, and considers network delays to be negligible, with sufficiently high intra-cloud network bandwidths.

C. *Components of Stratus*

Abstractions include the *Stratus master, Stratus node, Features, communication channels, and Policies*.

1) *Stratus Master*

The Stratus master: *(i)* maintains the latest information about Stratus nodes, *(ii)* builds optimized virtual platforms (VPs) as per user policies or to reject their construction, and *(iii)* manages virtual platforms through their lifecycles. These actions are performed via the common management channel.

2) *Stratus Node*

Stratus nodes contribute *Features* to each VP. New devices, i.e., nodes, export their *Features* to the master upon discovery and registration. Each node has its own a unique identification (ID).

3) *Features*

Each *Feature* describes a virtualized resource as the basic building block of a certain VP, for example, a processor or a display. A virtual platform (VP), therefore, can be described as a set of *Features* coordinated by the Stratus master. Each Stratus node must have at least one *Feature* identified with a unique ID. Each *Feature* has attributes describing its status and properties.

4) *Management Channel*

The management channel connecting Stratus nodes and master must be reliable, robust, and light-weight. Low overhead is particularly important for *thin* Stratus nodes with resource limitations like smartphones. The channel used in this paper is an improved implementation of the M-Channels described in [6].

5) *Event Forwarding*

Different *Features* of a VP can reside on different nodes, e.g., input devices of a VP may be on the Stratus node *(N)* while its CPU is on the Stratus node *(N+1)*. This requires the events raised on input devices to be forwarded to the CPU on a different node. Event forwarding is done via the Stratus management channel.

6) *User Policies*

A Policy describes the *Features* desired for a certain VP along with certain quantifiable *Feature* attributes. Policies can be defined dynamically and are used by sending them to the Stratus master. When master receives a policy, it reviews available resources (*Features*) and if sufficient resources (*Features*) exist, it establishes an appropriate VP by combining its available *Features*. Current policies appear in TABLE I. Fig. 2 depicts how Stratus components are connected with each other.

D. *Virtual Platform Use*

To understand how virtual platforms are established, consider an end user with a Stratus-enabled smartphone with HD recording functionality. The phone was used to record HD videos, which the user now wants to edit.

TABLE I. POLICY EXAMPLES

Policy	Description
High Performance	A performance oriented platform
Power Efficiency	The VP should minimize device cloud power usage, instead of maximizing performance.
Mobility	A user decides to leave a device cloud in which she operates, causing the master to migrate the appropriate VP to the mobile device being used.

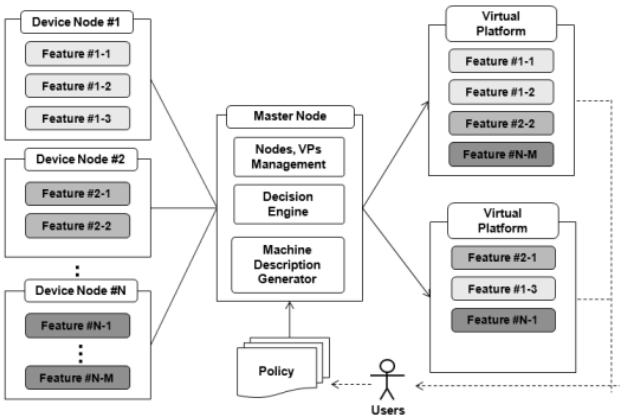

Figure 2. Relation between Stratus Components

1) Joining a Device Cloud

When the user enters the home, the location-based Stratus service in the phone permits it to join the home-based device cloud. The first step is to register the device at the Stratus master as a Stratus node. The device reports its *Features* and in addition, it receives from the Stratus master information about available *Features* in the device cloud it is joining, such as shared storage, for instance. Once it accepts the shared storage *Feature*, the user can freely access said storage in order to store the HD videos present in the phone. The user need not know where in the device cloud the shared storage is located.

2) Creation of Virtual Platform

As mentioned earlier, to conveniently edit HD videos present on a phone, a user requires a virtual platform with a large screen and a high performance CPU. The HD video editing policy stating these requirements is sent to the Stratus master, which then establishes the appropriate virtual platform using the devices present in the home. This policy states the following requirements; "*full size input devices*", "*large screen*", and "*HD editing software*". Given these requirements, the Stratus master decides whether the policy is able to be accepted or not, by inspecting the resource pool of the Stratus device cloud it manages. If accepted, the master sends a notification to the Stratus nodes that have the required *Features* in order to check whether they are still available. Such lazy availability checking serves to reduce intra-Stratus management communication.

Unlike the first two requirements in the HD video editing policy (*input devices* and *large screen*), the third requirement targets software. This means that the master must inspect shared storage for a virtual machine image with such software. If available, that image will be booted on the virtual platform established based on the first two requirements. Another option is to migrate to the phone's own virtual machine. In either case, the user does not need to know where the platform's CPU and memory are located. Once the VP has been successfully established, the input device and screen satisfying the policy are assigned into the VP by the Stratus master. From now on, the user is able to edit HD videos from the shared storage on the VP, using a large screen and full size keyboard rather than the small screen and touch screen interface present on the phone. A final step taken by the Stratus master is to update the status of *Features* in its resource pool.

3) Termination of virtual platform

If the user finishes video editing and wants to stop using the virtual platform, the master starts terminating it. This involves releasing the *Features* assigned to the virtual platform and suspending the platform's software state into a virtual machine image file on shared storage. There are two cases of VP termination: *(i)* normal termination is based on a request from the VP's user, and *(ii)* abnormal termination is caused by unexpected events like power failures or system halt of devices in the cloud. A watch-dog scheme detects abnormal terminations.

4) Leaving a Device Cloud

An entity can quit a device cloud explicitly by sending a quit notification to the master. Otherwise, an implicit departure is one in which the master does not receive a reply from a device within some time interval of checking the availability of *Features*. Those cause device removal from the cloud's resource pool which master has to handle.

III. EXPERIMENTAL EVALUATION

A potential advantage of device clouds is to provide to end users virtual platforms with inherently different or better properties than what can be provided by single machines. The experimental evaluations in this section demonstrate this advantage, and they also assess the overheads of VPs, both concerning their use and their construction.

A. Testbed

Our testbed is comprised of machines typically found in mobile and home computing systems:

- a quad core desktop with a high resolution (720p) display (Desktop #1 in TABLE II);
- a dual core desktop with an XGA screen (Desktop #2 in TABLE II);
- a netbook with a SWVGA screen and an Intel Atom processor, representing the mobile device in our settings (Mobile Device in TABLE II); and
- an NSF file server accessible by all home machines, which can store data as well as the virtual machine images being used.

Detailed device specifications appear in TABLE II. A netbook is used in place of an ARM-based handheld devices since its ISA is consistent with that of other machines. The netbook uses the Z5xx series Atom processors designed for battery-operated handheld devices.

For simplicity, the netbook is connected to other machines via a 100Mbit wired network rather than using its wireless connection. This bandwidth is in keeping with existing wireless connectivity available for home systems (e.g., consider that recent wireless standards can support up to 300Mbps bandwidth from IEEE 801.11n). All devices mount their root file system to the file server via NFS.

TABLE II. PARTICIPANTS OF THE DEVICE CLOUD

Devices	Desktop #1 (DT#1)	Desktop #2 (DT#2)	Mobile Device
Processor	Quad Core	Dual Core	Atom Z530
Memory	6GB	2GB	768MB
Storage	650GB	200GB	16GB
Network	1Gbit	1Gbit	100Mbit

There is one Stratus master (Desktop #1) and three Stratus nodes, jointly providing seven different types of *Features*: ***processor, memory, screen, keyboard, network, storage, mobility, and accelerator***. Each *Feature* defines its status with respect to its usefulness to cloud participants (e.g., how much storage others can use). Such status describes the feature's innate capability, whether the feature can be shared, whether it is present in a VP, as well as its current utilization. For this evaluation, a processor's innate capability is computed by multiplying its number of cores with its clock speed, a memory's capability as its size in megabytes, and a screen's capability as its vertical resolution, etc.

There are two *Features*: ***mobility and accelerator***. A Stratus node with a mobility *Feature* is considered as a mobile device by the Stratus master. An accelerator *Feature* means that the corresponding node has a special purpose hardware accelerator (e.g., encoding or decoding codec, 3D graphics). The accelerator *Feature* is under development.

B. Evaluation Results

There are many ways for users to gain advantages from using device clouds and their virtual platforms. This section evaluates the opportunities and overheads implied in device cloud use. The section is organized by presenting useful policies that driving platform composition.

1) High Performance Platforms

The purpose of the *High Performance Policy* in Stratus is to give users access to high end devices present in the cloud. As a motivating example, consider content captured on a portable device like a netbook that should be compressed before it is stored on disk. Using the netbook for that purpose will be slow, whereas a virtual platform that includes say, the home PC's fast processor, would perform much better.

To be more precise, to carry out such dynamically determined and computationally expensive tasks, several items of information are needed, including *(i)* which device in the cloud has the potential and *(ii)* the current resources to quickly complete such tasks, and *(iii)* how to move the user's dynamic state to where the computation is best performed. Further, such state movement should preserve the user's current working environment (e.g., personal settings), which implies that both data and working environment should be moved in unison.

The Stratus *High Performance Policy* achieves *(i)-(iii)* first by composing a high performance virtual platform based on current device availability and state and second by transparently migrating the user's computing environment to the platform's processing engine. In this case, the first step composes a VP that includes the home PC and the second step involves migrating the VM from the netbook to the home PC to carry out the tasks desired by end users. The platform composition is determined by maximizing the aggregate *Performance Index* of the nodes involved. For each node, this index is computed as (1):

$$\textit{Performance Index} = \textit{MemorySize} + \sum_{n=1}^{N} \{\textit{ClockFreqofCore}_n \times (1 - \textit{Util\_Core}_n)\} \quad (1)$$

** N is the number of the cores of processor in a device*

$ClockFreqofCore_n$ and $Util\_Core_n$ represent the clock frequency and the average of last five-minute utilization of the *n*-th core of a processor in a device. *MemorySize* represents the amount of device for a VP. The *Performance Index* used above is useful for certain classes of applications. For others, 3D games for instance, Stratus permits definition of alternative indices, such as GPU performance.

Note that with this policy, Stratus does not attempt to adjust the operating frequencies of participating nodes, the idea being that VP composition should be done in ways that preserve the local policies governing nodes' operation. Furthermore, users can explicitly state preferences for input/output devices, which means that the node that provides the screen *Feature* need not be the same as the node providing CPU and memory. The current implementation of Stratus attains this end by using a VNC-compatible protocol to forward the virtual screen of an established VP to whichever larger screen an end user wishes to use. Finally, all such actions are subject to admission control checks. For instance, if a node requests a maximum *Performance Index* not currently available in the cloud, such requests will be rejected. Similarly, if screen forwarding exceeds available network bandwidth, the composition of a VP requiring such an action will not be allowed. Stratus achieves this by actively monitoring capacity and bandwidth availability.

When the *High Performance Policy* is issued by the netbook device with the device cloud shown in TABLE II, the following actions are taken: *(i)* the Stratus master in DT#1 finds the largest screen and the node with the maximum *Performance Index*. Using the following description of *Feature* assignments for the VP - *<node#, Feature#>* - this results in a VP described as <1, 1>, <1, 2> , and <1, 3>. Stratus also prompts the user to select an input device (*keyboard*) among <1, 4>, <2, 4>. The final VP consists of four virtual cores, 512MB memory, a 720P screen, and a full-size keyboard. To use it, Stratus performs live migration of the user's virtual machine image from the netbook to the composed VP.

VP composition for the small-scale systems present in homes occurs with latencies not noticeable to end users. The current version of Stratus (not fully optimized) takes 5,004μs to find the *Features* for the high performance VP, and the live migration to the VP took 15,451ms for the VM image present on the netbook. In Section III.C, we discuss how much time is consumed by each stage during establishing the VP.

At the same time, substantial performance improvements can result from using the newly constructed VP vs. continued operation on the netbook. These are evident when uncompressing a large size file (bzip2 compressed file system image, the size is 4.4GB) using 7zip with multithread option (-mmt), and transcoding a video file (360MB 720P resolution from mpeg4 into H.264). As shown in TABLE III the high performance platform conducts these tasks in 7.9% for uncompressing and 11.25% for transcoding of the time compared to running these on the netbook. In other words, the overheads of platform composition and VM migration are almost amortized by this single high end decompression task. A final interesting note is that the performance policy will always prefer using DT#1 because of its vastly superior performance index, unless of course, that machine is already heavily used by other VPs present in the system (resulting in its *Performance Index* being low). In such cases, machines like DT#2 will be selected.

2) Reducing Power Consumption in Device Cloud

Stratus can automatically generate a power-efficient VP, without requiring users to understand device power usage characteristics. This is initiated by the user's activation of the *Power Efficiency Policy*. This policy uses the *Performance Index* maintained by Stratus to assess node and device power consumption, thus making the simplifying assumption that the index is proportional to the voltage and clock-speeds of its processors[11], and memory power consumption. The Stratus decision engine uses the following procedure to compose low power VPs:

- first, find the node with the lowest maximum performance index in the device cloud;
- confirm that the utilization of that node's processor *Feature* is less than 70%[1] and if yes, assign it as the representative node for the new VP; if not, find the second best node; etc.

Consider the following case to evaluate this policy. Suppose a user wants a VP that plays SVGA content from a steaming service but to do so with the *Power Efficiency Policy*. To emulate a streaming service like YouTube or Hulu.com, we add a streaming server to the testbed, and activate the policy at a time when all nodes in the cloud are in idle states. Then Xen hypervisor on each node runs its local on-demand power management option. The outcome of the experiment is shown in TABLE IV, which depicts the comparative power consumption of the *Performance* and *Power Efficiency* policies. It is evident from these results that VPs can be composed so as to obtain reduced power without causing performance problems for users, such as frame dropping. The video file for this evaluation is encoded via XviD codec with SVGA resolution. TABLE V depicts additional results when using 5 VPs with the same video workload, to simulate a multi-user scenario. In the first case, in TABLE IV a 3% power savings is obtained, whereas the second case in TABLE V shows a 10.5% improvement in power usage. The second case shows increased frame drops, but at a low level barely noticeable by end users.

[1] The utilization of 70% is an empirical value.

TABLE III. PERFORMANCE COMPARISON:VP VS. NETBOOK

Performace	Netbook	Virtual Platform
Decompressing Time (Sec)	4845	383
Transcoding Time (Sec)	12312	1386

TABLE IV. POWER COMSUMPTION COMPARISON - SINGLE VP

Policy	Idle	Power Efficiency	Performance
Power Consumption (watt)	177.2	178.3	183.4
Drop Frames/Total (drop-rate%)	-	0/34631 (0%)	0/34631 (0%)

TABLE V. POWER COMSUMPTION COMPARISON - 5 VPS

Policy	Power Efficiency	Performance
Power Consumption (W)	201.5	225.2
Drop Frames/Total (drop-rate%)	9/34631 (≈0%)	3/34631 (≈0%)

Further, VPs on same node have to share a screen *Feature* since there are only 3 screens in this device cloud. Finally, although DT#1 could support additional VPs in terms of its computational power, it has insufficient memory to host 5 VPs, causing the 5th VP to be established on DT#2 as shown in TABLE VI.

There are several reasons for the relatively small power savings obtained with this policy. First, the policy only considers processors and memory, ignoring other hardware like graphics cards. Second, there is currently no runtime re-configuration, to replace power hungry nodes with power-efficient ones used by existing VPs (e.g., at times when power-efficient systems were not available). The earlier work in [10] presents methods with which such dynamic reconfiguration can be done. Third, in order to avoid situations in which power efficient policies create VPs with insufficient performance, our current implementation asks users to provide information about the kind of work VPs will be asked to perform, such as *OFFICE SUITES, WEB BROWSING, PLAYING CONTENTS, OTHER*, where a request labeled as *OTHERS* is not accepted by a power efficient VP. In summary, this case demonstrates the potential for energy savings with Stratus, but additional work is needed to create a robust energy savings policy and implementation.

3) Enhanced Battery Life of Devices in Device Cloud

A straightforward idea is to offload computing from mobile to stationary devices, whenever possible. Such offloading is done by the *Long Lasting Battery Policy*. When issued by mobile devices, this policy attempts to maximally extend their battery lives, by first creating a new VP with the same or better *Performance Index* as the existing one and then migrating the virtual machine image to the established VP. The mobile device, then, merely acts as an input and output device, permitting its processors to operate at lower frequencies.

TABLE VI. DISTRIBUTION OF VPS BASED ON DIFFERENT POLICIES

Policy	Power Efficiency	Performance
DT#1	0	4
DT#2	3	1
Netbook	2	0

TABLE VII. COMPARISON: BATTERY LASTING TIME

Environment	Netbook	Battery Policy
Battery Lasting Time (min)	92.8	124
Power Consumption (W)	16.2	13.3

We evaluate the utility of this policy by playing VGA resolution movie clips continuously until the battery of the netbook becomes completely discharged. As shown in TABLE VII we can achieve almost 18% less power consumption and 15% more operation time with almost the same number of frame-drops.

4) Enabling Virtual Platform Mobility

The *Mobility Policy* permits users to continue with mobility whatever actions they have been taking within some home or office device cloud. This is done by first establishing a new VP on the node with battery and wireless network *Features* and then moving the virtual machine image from the node issuing the policy to new mobile VP. When multiple mobile nodes are present, we now prompt the user, but other means like accelerator-based sensing of pick up actions could be used. At present, when prompting users, Stratus provides several useful items of information, including remaining battery capacity, estimated battery operation time, and maximum *Performance Index*.

C. Overhead

Establishing a VP involves several steps.

S1. Requesting a VP: a node in a device cloud issues a request for a new VP to the master.

S2. Reporting status to the master: the master asks all participants in a device cloud to provide information about their *Features*.

S3. Exploiting the decision engine and generating machine description: after gathering information from all nodes, the master tries to allocate the best *Features* for the VP being constructed, as per the user policy being used.

S4. VP notification: the machine description is sent to the representative node of the VP being established.

TABLE VIII shows the time consumed for each stage when the *High Performance Policy* is issued in the testbed shown in Section III.A. Additional results shown TABLE IX establish that VP construction overheads are almost identical for VPs with larger numbers of nodes (i.e., 5 in this case). These measurements show that total overheads are moderate (about 5ms) and that with these overheads, even the current Stratus solution will likely scale to the ~100 devices expected in homes or small office systems.

TABLE VIII. DELAYS IN ESTABLISHING A HIGH PERFORMANCE VP

Step	S1	S2	S3	S4	Misc.	Total
Delay (*us*)	51	4786	136	11	20	5004

TABLE IX. CONSTRUCTION OVERHEAD (LARGER NUMBERS OF NODES)

Numbers of Nodes	2	3	4	5
Delays (*us*)	5084	5004	5126	5296

Alternative implementations and additional optimizations are needed for larger configurations; these include approaches in which device clouds are zoned with zone masters relating to each other as peers[10], of pre-constructed VPs, push-based updates to information about *Features*, etc.

IV. RELATED WORK

A. Seamless User Experience based on Virtualization

Prior work has used virtualization technologies to provide users with seamless computing environments. ISR[12], the Collective project[13], SoulPad[14], Keychain[15], Spirits[18], and Cloudlets[20] store all of a user's computing environment and state in virtual machines, using VM migration to move VMs to the device the user wants to use. VMs can be migrated through a network connection, via storage devices like USB disks, and they may be assisted by nearby resources so as to avoid the potential high overheads arising from accessing remote cloud resources[18][20].

All of these approaches target single physical platforms, however, in contrast to the Stratus approach in which multiple platforms' devices are utilized to gain improved end user experiences. Also missing from such work is the automation via user policies shown in our work.

Building on previous work like thin-client computing, an alternative model offered by cloud computing is to rely on remote server systems to provide needed functionality like Amazon EC2. Device clouds simply extend those models to also exploit locally available devices in addition to using remote services. This may also help deal with the privacy and security issued raised for cloud systems, by differentially using local vs. remote service capabilities.

B. Resource Allocation in Distributed Systems

Stratus leverages well-established techniques for managing the resources of distributed and media systems[16][17][22]. There remain many open issues with its current implementation, however, including those pertaining to dynamic platform reconfiguration, the use of richer and thus, more capable formulations of policies, and additional work on runtime management to provide virtual platforms that are more robust and/or capable in service offerings compared to the single platforms now used by end users. The HTML 5 device APIs enable web applications to access each device in low level.

V. Conclusions and Future Work

Device clouds give rise to virtual platforms that provide end user services superior to those offered by single physical devices. Advantages include higher performance, better availability in lieu of say, improved end user experiences. These properties are due to the elastic and dynamic nature of virtual platforms, driven by policies encoding end user needs. Stratus is evaluated with several policies useful in the home or office settings targeted by device clouds, with experimental results establishing the relative advantages of virtual vs. physical platforms as well as the overheads implied in VP construction and use.

There are extensive opportunities for future work. The current implementation of device clouds has not been integrated with remote cloud infrastructures. Specifically, of interest to the media-centric examples studied in our work is improved flexibility in content delivery, to obtain content from both local (e.g., the home PC disk) and remote sources. Many factors can be considered to achieve such integration.(e.g. cost[21]) Another direction is to exploit the VM-based encapsulation to address security and privacy concerns when a user's VM is employed outside her home.

References

[1] Dearman, D. and Pierce, J. S. "It's on my other computer!: computing with multiple devices," In Proceeding of the Twenty-Sixth Annual SIGCHI Conference on Human Factors in Computing Systems. CHI '08. ACM, New York, NY, 767-776.

[2] Pierce, J. S. and Nichols, J. "An infrastructure for extending applications' user experiences across multiple personal devices," In Proceedings of the 21st Annual ACM Symposium on User interface Software and Technology. UIST '08. ACM, New York, NY, 101-110.

[3] Oulasvirta, A. and Sumari, L. "Mobile kits and laptop trays: managing multiple devices in mobile information work," In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '07. ACM, New York, NY, 1127-1136.

[4] Song, X. and Ramachandran, U. "MobiGo: A Middleware for Seamless Mobility," In Proceedings of the 13th IEEE international Conference on Embedded and Real-Time Computing Systems and Applications. RTCSA. IEEE Computer Society, Washington, DC, 249-256.

[5] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. "Xen and the art of virtualization," In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles. SOSP '03. ACM, New York, NY, 164-177.

[6] Kumar, S., Talwar, V., Ranganathan, P., and Ripal Nathuji, K. S. "M-channels and m-brokers: Coordinated management in virtualized systems," In Workshop on Managed Multi-Core Systems (June 2008)

[7] Padala, P., Hou, K., Shin, K. G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., and Merchant, A. "Automated control of multiple virtualized resources," In Proceedings of the 4th ACM European Conference on Computer Systems. EuroSys '09. ACM, New York, NY

[8] Soror, A. A., Minhas, U. F., Aboulnaga, A., Salem, K., Kokosielis, P., and Kamath, S. "Automatic virtual machine configuration for database workloads," ACM Trans. Database Syst. 35, 1 (Feb. 2010), 1-47.

[9] Rolia, J., Cherkasova, L., Arlitt, M., and Andrzejak, A. "A capacity management service for resource pools," In Proceedings of the 5th international Workshop on Software and Performance. WOSP '05. ACM, New York, NY, 229-237.

[10] Sanjay Kumar, Vainsh Talwar, Vibhore Kumar, Partha Ranganathan, Karsten Schwan. "vManage: Loosely Coupled Platform and Virtualization Management in Data Centers," In 6th International Conference on Autonomic Computing & Communications (ICAC), Barcelona, Spain, 2009.

[11] Pillai, P. and Shin, K. G. 2001. "Real-time dynamic voltage scaling for low-power embedded operating systems," In Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles. SOSP '01. ACM, New York, NY, 89-102.

[12] Kozuch, M. and Satyanarayanan, M. 2002. "Internet Suspend/Resume," In Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications. WMCSA. IEEE Computer Society, Washington, DC, 40.

[13] Chandra, R., Zeldovich, N., Sapuntzakis, C., and Lam, M. S. "The collective: a cache-based system management architecture," In Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2. USENIX Association, Berkeley, CA, 259-272.

[14] Cáceres, R., Carter, C., Narayanaswami, C., and Raghunath, M. "Reincarnating PCs with portable SoulPads," In Proceedings of the 3rd international Conference on Mobile Systems, Applications, and Services.MobiSys '05. ACM, New York, NY, 65-78

[15] Muthukarrupan Annamalai, Andrew Birrell, Dennis Fetterly, and Ted Wobber, "Implementing Portable Desktops: a New Option and Comparisons," no. TR-2006-151, October 2006, Microsoft Corporation

[16] Poellabauer, C., Abbasi, H., and Schwan, K. 2002. "Cooperative run-time management of adaptive applications and distributed resources," In Proceedings of the Tenth ACM international Conference on. MULTIMEDIA '02. ACM, New York, NY, 402-411.

[17] Xu, D., Wichadakul, D., and Nahrstedt, K. 2000. "Multimedia Service Configuration and Reservation in Heterogeneous Environments," In Proceedings of the the 20th international Conference on Distributed Computing Systems ICDCS 2000. IEEE Computer Society, Washington, DC, 512.

[18] Raj, H., Seshasayee, B., O'Hara, K., Nathuji, R., Schwan, K., and Balch, T. 2006. "Spirits: Using Virtualization and Pervasiveness to Manage Mobile Robot Software Systems," In Proceedings of the Second IEEE International Workshop on Self-Managed Networks, Systems & Services (SelfMan, in conjunction with ICAC). June 2006

[19] Kannan, S., Badu, K., Gavrilovska, A., and Schwan, K., "VStore++: Virtual Storage Services for Mobile Devices," In Proceedings of International Workshop on Mobile Computing and Clouds (Oct 28, 2010) MobiCloud 2010, ICST, Santa Clara, CA

[20] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. 2009. "The Case for VM-Based Cloudlets in Mobile Computing," IEEE Pervasive Computing 8, 4, 14-23.

[21] Pandey, S., Barker, A., Gupta, K.K., Buyya, R. "Minimizing Execution Costs when Using Globally Distributed Cloud Services,", 24th IEEE International Conference on Advanced Information Networking and Applications, 2010, 222-229

[22] Hyunjoo Kim, Yaakoub el-Khamra, Shantenu Jha, and Manish Parashar."Exploring application and infrastructure adaptation on hybrid grid-cloud infrastructure," In Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10). ACM, New York, NY, USA, 402-412