iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 2 **Issue:** IX **Month of publication:** September 2014

DOI:

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

ISSN: 2321-9653

INTERNATIONAL JOURNAL FOR RESEARCH IN APPLIED SCIENCE AND ENGINEERING TECHNOLOGY (IJRASET)

Energy Management in Mobile Devices with the Cinder Operating System

Anjali Agrawal[1], Aashima Randhawa[2], Karan Bhalla[3]

[1,2,3]*Dronacharya College Of Engineering, Gurgaon*

Abstract: ***We present Cinder, an operating system for mobile phones and devices, that allows users and applications to control and manage limited device resources such as energy. We argue that controlling energy allocation is an important and increasingly useful feature for operating systems, especially on mobile devices. We present two new low-level abstractions in the Cinder operating system, taps and reserves, which store and distribute energy for application use. We identify three key properties of control isolation, delegation, and subdivision and show how using these abstractions can achieve them. We also show how the architecture of the HiStar information flow control kernel lends itself well to control energy. We prototype and evaluate Cinder on a popular smart phone, the Android G1.***

Keyword: Energy, mobile phones, power management.

I. INTRODUCTION

In the past decade, mobile phones have emerged as a dominant computing platform for end users. These very personal computers depend heavily on graphical user interfaces, always-on connectivity, and long battery life, yet in essence run operating systems originally designed for workstations (Mac OS X/Mach) or time-sharing systems (Linux/Unix). Historically, operating systems have had poor energy management and accounting. This is not surprising, as their APIs standardized before energy was an issue. This limited control and visibility of energy is especially problematic for mobile phones, where energy and power define system lifetime. In the past decade, phones have evolved from low-function proprietary applications to robust multi- programmed systems with applications from thousands of sources. Apple announced that as their App Store houses 185,000 apps for the iPhone with more than 4 billion application downloads. This shift away from single-vendor software to complex application platforms means that the phone's software must provide effective mechanisms to manage and control energy as a resource. Such control will be even more important as the danger grows from buggy or poorly designed applications to potentially malicious ones.

In the past year, mobile phone operating systems began providing better support for understanding system energy use. Android, for example, added a UI that estimates application energy consumption with system call and event instrumentation, such as processor scheduling and packet counts. This is a step forward, helping users understand the mysteries of mobile device lifetime. This paper presents Cinder, a new operating system de- signed for mobile phones and other energy-constrained computing devices. Cinder extends the HiStar secure kernel to provide new abstractions for controlling and accounting for energy: reserves and taps. Taps and reserves compose together to allow applications to express their intentions, enabling policy enforcement by the operating system.

Cinder estimates energy consumption using standard device-level accounting and modeling. HiStar's explicit information flow control allows Cinder to track which parties are responsible for resource use. Cinder is the first research operating system that runs on a mobile phone.

This paper makes three research contributions. Firstly, it proposes reserves and taps as new operating system mechanisms for managing and controlling energy consumption. Secondly, it describes experiences in writing a mobile phone operating system, out- lining the challenges and impediments faced when conducting systems research on the dominant end-user computing platform of this decade. Thirdly, it evaluates the effectiveness and power of these mechanisms in a variety of realistic and complex application scenarios running on a real mobile phone.

II. A CASE FOR ENERGY CONTROL

This section motivates the need for low-level, fine-grained energy control in a mobile device operating system. It starts by reviewing some of the prior work on energy visibility and the few examples of coarse energy control. The next section describes reserves and taps, abstractions which provide these mechanisms at a fine granularity.

There is rich prior work on addressing the visibility problem of attributing consumption to application principals. Control, in contrast, has seen much less effort. Early systems like EcoSystem proposed high level application power limits. Mobile applications today, however, are much more complex: they spawn and invoke other services and have a much richer set of peripherals to manage. We believe that for users and applications to effectively control power, an operating system must provide three mechanisms: isolation, subdivision, and delegation. We motivate these mechanisms through three application examples that we follow through the rest of the paper. Isolation is a fundamental part of an operating system. Memory and IPC isolation provide security, while cpu and disk space isolation ensure that processes cannot starve others by hogging needed resources. As a more concrete example, the energy a phone reserves for an emergency 911 call should be isolated from the rest of the system, so that other programs cannot use it. Web browsers run (sometimes untrusted) plugins. Given that a browser receives a finite amount of power, it might want to protect itself from buggy or poorly written plugins that waste CPU energy. The browser would like to subdivide its energy so that it can give plugins a small fraction, knowing that isolation will prevent them from using its own lion's share. The ability to subdivide energy is critical for applications to be able to invoke other services without sacrificing all of their own resources. Finally, there are times when applications need to allow others to use their energy, but do not want to carve off a reserved, isolated subdivision. The ability to delegate resources is an important enabler of inter-application cooperation. For example, the Cinder networking stack implicitly transfers energy into a common radio activation pool when an application cannot afford the high initial expense of powering up the radio. By delegating their energy to the radio, multiple processes can contribute to expensive operations; this can not only improve quality of service, but even reduce energy consumption. Prior systems like EcoSystem and Currency provide isolation, but not subdivision or delegation. Isolation is sufficient when applications are static entities, but not when they themselves spawn new processes or invoke complex services. Subdivision lends naturally to standard abstractions such as process trees, resource containers and quotas, while delegation is a kin to priority inheritance.

III. DESIGN

Cinder is based on the HiStar operating system which is a secure exo-kernel that controls information flow using a label mechanism. The kernel provides a small set of kernel object types to applications, from which the rest of system is built: threads, address spaces, segments, gates, containers, and devices. Cinder adds two new kernel object types: reserves and taps. This section gives an overview of HiStar, describes reserves and taps, gives examples of how they can be used, and provides details on their security and information flow.

HiStar

HiStar is composed of six first-class kernel objects, all protected by a security label. Its segments, threads, address spaces, and devices are similar to those of conventional kernels. Containers enable hierarchical control over deallocation of kernel objects – objects must be referenced by a container or face garbage collection. Gates provide protected control transfer of a thread from one address space to a named offset in another; they are the basis for all IPC.

Reserves

A reserve describes a right to use a given quantity of a resource, such as energy. When an application consumes a resource the Cinder kernel reduces the values in the corre- sponding reserve. The kernel prevents threads from performing actions for which their reserves do not have sufficient resources. Reserves, like all other kernel objects, are protected by a security label (§3.5) that

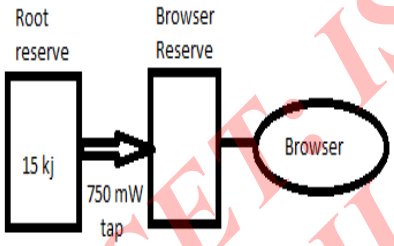
controls which threads can ob- serve, use, and manipulate it. Reserves allow threads to delegate and subdivide re- sources. Reserves also provide accounting by tracking application resource consumption. Applications may access this accounting information in order to provide energy-aware features. Finally, reserves can be deleted directly or indirectly when some ancestor of their container is deleted, just as a file can be deleted either directly or indirectly when a directory containing it is deleted in a UNIX system.

Taps

A tap transfers a fixed quantity of resources between two reserves per unit time, which controls the maximum rate at which a resource can be consumed. For example, an appli- cation reserve may be connected to the system battery via a tap supplying 1 mJ/s (1 mW).

Taps aid in subdividing resources between applications since partitioning fixed quantities is impractical for most policies. A user may want her phone to last at least 5 hours if she is surfing the web; the amount of energy the browser should receive is relative to the length of time it is used. Providing resources as a rate naturally addresses this. Taps are made up of four pieces of state: a rate, a source reserve, a sink reserve, and a security label containing the privileges necessary to transfer the resources between the source and sink

Figure:



A 15 kJ battery, or root reserve, connected to a reserve via a tap. The battery is protected from being misused by the web browser. The web browser draws energy from an isolated reserve which is fed by a 750 mW tap.

Access Control & SecurityAny thread can create and share reserves or taps to subdivide and delegate its resources. This ability introduces a problem of fine grained access control. To solve this, reserves and taps are protected by a security label, like all other kernel objects. The label describes the privileges needed to observe, modify, and use the reserve or tap. Since a tap actively moves resources between a source and sink reserve, it needs privileges to observe and modify both reserve levels; to aid with this, taps can have privileges embedded in them.

IV. CINDER ON THE HTC DREAM

Controlling energy requires measuring or estimating its consumption. This section describes Cinder's implementation and its energy model. The Cinder kernel runs on AMD64, i386, and ARM architectures. All source code is freely avail- able under open-source licenses. Our principal experimental platform is the HTC Dream (Google G1), a modern smart- phone based on the Qualcomm MSM7201A chipset.

Energy accounting

Energy accounting on the HTC Dream is difficult due to the closed nature of its hardware. It has a two-processor design. The operating system and applications run on an ARM11 processor. A secure, closed ARM9 coprocessor manages the most energy hungry, dynamic, and informative components (e.g. GPS, radio, and battery sensors). The ARM9, for example, exposes the battery level as an integer from 0 to 100. Recent work on processors has shown that fine-grained performance counters can enable accurate energy estimates within a few percent [Economou 2006; Snowdon 2009]. Without access to such state in the HTC Dream, however, Cinder relies on the simpler well-tested technique of building a model from offline-measurements of device power states in a controlled setting [Flinn 1999b; Fonseca 2008; Zeng 2002]. Phones today use this approach, and so Cinder has equivalent accuracy to commodity systems.

Power Model

Our energy model uses device states and their duration to estimate energy consumption. We measured the Dream's energy consumption during various states and operations. All measurements were taken using an Agilent Technologies E3644A, a DC power supply with a current sense resistor that can be sampled remotely via an RS-232 interface. We sampled both voltage and current approximately every 200 ms, and aggregated our results from this data. While idling in Cinder, the

Dream uses about 699 mW and another 555 mW when the backlight is on. Spinning the CPU increases consumption by 137 mW. Memory-intensive instruction streams increase CPU power draw by 13% over a simple arithmetic loop. However, the HTC Dream does not have hardware support to estimate what percentage of instructions are memory accesses. The ARM processor also lacks a floating point unit, leaving us with only integer, control flow, and memory instructions. For these reasons, our CPU model currently does not take instruction mix into account and assumes the worst case power draw (all memory intensive operations).

Peripheral Power

The baseline cost of activating the radio is exceptionally high: small isolated transfers are about 1000 times more expensive, per byte, than large transfers. Results demonstrate that the overhead involved dominates the total power cost for flows lasting less than 10 seconds in duration, regardless of the bitrate. An application powers up the radio by sending a single 1-byte UDP packet. The secure ARM9 automatically returns to a low power mode after 20 seconds of inactivity. Because the ARM9 is closed, Cinder cannot change this inactivity timeout. With this workload, it costs 9.5 joules to send a single byte! One lesson from this is that coordinating applications to amortize energy start-up costs could greatly improve energy efficiency. In x5.5 we demonstrate how Cinder can use reserves and taps for exactly this purpose.

Mobility & Power Model Improvements

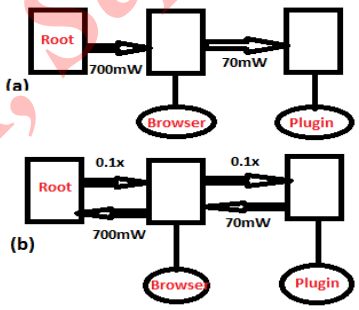
Cinder's aim is to leverage advances in energy accounting to allow users and applications to provision and manage their limited budgets. Accurate energy accounting is an orthogonal and active area of research. Cinder is adaptable and can take advantage of new accounting techniques
or information exposed by device manufacturers.

V. APPLICATIONS

To gain experience with Cinder's abstractions, we developed applications using reserves and taps.

Energy wrap

Taking advantage of the composability of Cinder's resource graph, the energy wrap utility allows any application to be sandboxed even if it is buggy or malicious. Energywrap takes a rate limit and a path to an application binary. The utility creates a new reserve and attaches it to the reserve in which energywrap started by a tap with the rate given as input. After forking, energywrap begins drawing resources from the newly allocated reserve rather than the original re- serve of the parent process and executes the specified program. This allows even energy-unaware applications to be augmented with energy policies. energywrap has proved useful in implementing policies while designing and testing Cinder, particularly for legacy applications that have no notion of reserves or taps. Since energywrap runs an arbitrary executable, it is possible to use energywrap to wrap itself or shell scripts, which may invoke energywrap with other scripts or applications. allows a wide class of ad hoc policies to be scripted using standard shell scripting or on-the-fly at the command line.
Figure.



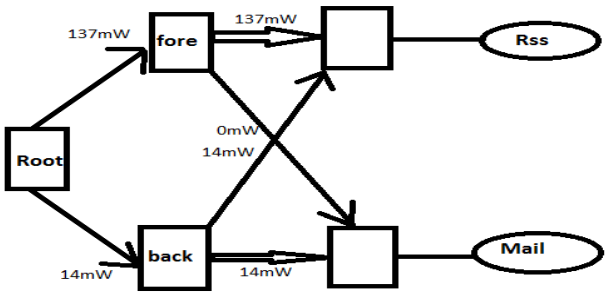
(a) A web browser configured to run for at least 6 hours on a 15 kJ battery. The web browser further ensures that its plugin cannot use more than 10% of its energy.
(b) (b) Adding 0.1x backward proportional taps promotes sharing of excess energy unused by the browser and plugin.

Energy-Aware Applications

Using Cinder, developers can gain fine grained control of resources within their applications, providing a better experience to end users. This includes adaptive policies for pro- grams where partial or degraded results are still useful, andoffer a compromise between battery life and user experience. For example, smart applications may scale the quality of streaming

video or reduce texture quality in a game when available energy is low, since the user can still watch a video or play a game when insufficient resources are available to run at full fidelity.

Figure.



RSS is running in the foreground so the task manager has set its tap to give it additional power. Mail is running in the background, and can only draw energy from the background reserve. This ensures that actual battery consumption matches the user's expectation that the visible application is responsible for most energy consumption.

VI. EXPERIENCE DEVELOPING ON A MOBILE PLATFORM

We ported Cinder to the HTC Dream mobile phone. Because developing a kernel for a mobile phone platform is a nontrivial task that is rarely attempted, we describe our process here in detail. To run Cinder on the HTC Dream, we first ported the kernel to the generic ARM architecture (2,380 additional lines of C and assembly).MSM7201A-specific kernel device support for timers, serial ports, framebuffer, interrupts, GPIO pins, and keypad required another 1,690 lines of C. Cinder implements the GSM/GPRS/EDGE radio functionality in userspace with Android driver ports. Implementing radio functionality is particularly difficult, as it requires access to secure and undocumented hardware that is not directly accessible from the processor. For instance, the MSM7201A chipset includes two cores: the ARM11 runs application code (Cinder), while a secure ARM9 controls the radio and other sensitive features. Accessing these features requires communicating between the cores using a combination of shared memory and interrupt lines. We first mapped the shared memory segment into a privileged user-level process and ported the Android Linux kernel's shared memory device to userspace .This daemon, smdd (4,756 lines), exports ARM9 services via gate calls to other consumers, including the radio interface library (RIL). The RIL generates and consumes messages between cores that initiate and respond to radio events, such as dialing a number or being notified of an incoming call. In Android, the radio interface library consists of two parts: an open source generic interface library that provides common radio functions across different hardware platforms, and a device-specific, Android-centric shared object that interfaces with specific modem hardware (libril.so). Unfortunately, libril.so is closed-source and precompiled for Android: this makes it excessively difficult to incorporate into another operating system. Without hardware documentation or tremendous reverse engineering, using the radio requires running this shared object in Cinder. To do so, we wrote a compatibility shim layer to emulate both Android's "bionic" libc interface, as well as the various /dev devices it normally uses to talk to the ARM9 (1,302 lines of C). We rewrote the library's symbol table to link against our compatibility calls, rather than the binary-incompatible uClibc functions and syscalls that regular Cinder applications use. Finally, we wrote a port of the radio interface library frontend that provides gates to service radio requests from applications needing network access. Cinder currently supports the radio data path (IP), and can send and receive SMS text messages. Cinder can also initiate and receive voice calls, but as it does not yet have a port of the audio library, calls are silent. In retrospect, since hardware documentation is unavailable, basing our solution on Android, rather than HiStar, would have been far simpler from a device support perspective. Crucially, however, our implementation atop Linux trades the simple and accurate IPC resource accounting needed in energy management for device drivers (x7.1).We felt that a cleaner slate justified the additional tedium as well as the reduced hardware support present in our prototype. In summary, even trivial radio operation is quite complicated, requiring about 12,000 lines of userspace code along with the 263 KiB closed libril.so. In comparison, the entire Cinder kernel consists of about 27,000 lines of C for all four CPU architectures and all device drivers. The kernel is only 644 KiB – less than 2.5 times the size of libril.so.

Cinder-Linux versus Cinder-HiStar

ISSN: 2321-9653

Cinder was initially implemented on HiStar because several key behaviors of the platform are naturally expressible using HiStar's abstractions. One such feature of Cinder is resource delegation between principals. Consider a common situation where a client process P requires work to be performed on its behalf by a daemon process D. A real world example is the radio interface layer daemon on the Android platform. Cinder must ensure P is charged for any work D performs on its behalf – or, equivalently, it must ensure that P provides the resources that D's code uses to run. HiStar's abstractions achieve this behavior cleanly and simply. A process in HiStar is a container, containing an address space and one or more threads. IPC is performed through special gates defined by the process – a thread belonging to process P can enter a gate defined by process D, after which the thread has access to D's address space, though while under control of D's code text. When process P requires service from daemon D a thread, T, belonging to P enters D's address space via a gate. Cinder debits T for work it performs as usual even though it executes under the control of D's code, correctly billing consumption to P. This way, HiStar's IPC mechanism easily achieves the desired delegation behavior. Linux, on the other hand, uses several different facilities to provide IPC, many of which are based upon message passing between processes. A few examples are pipes, Unix domain sockets, message queues, and semaphores. These forms of IPC occur without any resource sharing or attribution between processes. This subverts delegation since process P may elicit work by daemon D on its behalf without providing the resources for the work. To compensate, Linux needs to verify that the calling process has provided adequate resources to perform the desired request. However, existing IPC mechanisms in Linux are not built with the goal of discovering the identity of the caller in mind. Consider a daemon D that reads requests from a named pipe in the filesystem. When D reads from the pipe, it only knows the writing process has permissions to access the pipe. In general, it cannot identify which process in the system made the request, and thus does not know which process to debit. To mitigate this problem, Cinder-Linux needs a way for the daemon to determine the identity of the calling process. One possibility is to have a user level protocol in which a calling process P encodes both its identity and a description of how D can access resources that P has set aside for D within the request. For example, it could format a request as a triple: hpid, reserve id, requesti. D accesses the reserve named in the request, and only performs work once it ensures the caller has provided sufficient resources in payment. Since a user level process can lie abouts it credentials, the protocol is not robust against malicious applications. A more robust mechanism would require new kernel IPC mechanisms. Both Cinder-HiStar and Cinder-Linux must prevent resource misuse. In particular, D must not co-opt P's resources for performing unrelated tasks, and process P must provide resources for work performed by D on its behalf. Providing these guarantees on Linux requires either a fine grained permissions system or, alternatively, some form of information flow control or tracking (with which the daemon could determine which process sent a given request). In contrast, HiStar's existing information flow control mechanisms easily provide the necessary protection. Linux has the benefit of being an established operating system with vast device driver support and the entire Android platform. As a result, it is easier to write real-world applications. Consequently, we have written an initial implementation of Cinder that runs on top of the Linux and the Android platform on the Dream. The basics of Cinder-Linux remain the same as Cinder-HiStar aside from resource attribution issues for IPC and fine-grained permissions. Most implementation of the Cinder abstractions are independent of the underlying operating system and similar on HiStar and Linux. Some differences in the implementation do exist, however. For example, Cinder-HiStar flows taps during scheduler timer interrupts, while Cinder-Linux uses a kernel thread. One area of future work is further testing the concepts and features of Cinder on the Cinder-Linux platform.

VII. RELATED WORK

We group related work into three categories: resource management, energy accounting, and energy efficiency.

Resource Management

Cinder's taps and reserves build on the abstraction of resource containers [Banga 1999]. Like resource containers, they provide a platform for attributing resource consumption to a specific principal. By separating resource management into rates and quantities, however, Cinder allows applications to delegate with reserves, yet reclaim unused resources. This separation also makes policy decisions much easier. Since resource containers serve both as limits and reservations, hierarchical composition either requires a single policy (limit or reserve) or ad hoc rules (a guaranteed CPU slot cannot be the child of a CPU usage

limit). Linux has recently incorporated "cgroups" [Menage 2008] into the mainline kernel, which are similar to resource containers, but group processes rather than threads. They are hierarchical and rely on "subsystem" modules that schedule particular resources (CPU time, CPU cores, memory). ECOSystem [Zeng 2002, 2003] presents an abstraction for energy, "currentcy", which unifies a system's device power states. It represents logical tasks using a flat form of resource containers [Banga 1999] by grouping related processes in the same container. This flat approach makes it impossible for an application to delegate, as it must either share its container with a child or put it in a new container that competes for resources. Like ECOSystem, Cinder estimates energy consumption with a software-based model that ties runtime power states to power draw. ECOSystem achieves pooling similar to Cinder's netd for devices with non-linear power consumption (disk and network access), using unique cost models for each device. Cinder simplifies construction of these policies using its fine-grained protection mechanism and reserves to provide the same result in userspace.

Measurement, Modeling, and Accounting

Accurately estimating a device's energy consumption is an ongoing area of research. Early systems, such as ECOSystem [Zeng 2002], use a simple linear combination of device states. Most modern phone operating systems, such as Symbian and OS X, follow this approach. PowerScope improves CPU energy accuracy by correlating instrumented traces of basic blocks with program execution [Flinn 1999b]. A more recent system, Koala, explores how modern architectures can have counter-intuitive energy/performance tradeoffs, presenting a model based on performance counters and other state [Snowdon 2009]. A Koala-enabled system can use these estimates to specify a range of policies, including minimizing energy, maximizing performance, and minimizing the energy-delay product. The Mantis system achieves similar measurement accuracy to Koala using CPU performance counters [Economou 2006]. Quanto [Fonseca 2008] extends the TinyOS operating system to support fine-grained energy accounting across activities. Using a custom measurement circuit, Quanto generates an energy model of a device and its peripherals using a linear regression of power measurements. By monitoring the power state of each peripheral and dynamically tracking which activity is active, Quanto can give precise breakdowns of where a device is spending energy. PowerBooter and PowerTutor [Zhang 2010] explore the generation of detailed power models for a full-featured smartphone (the HTC Magic) providing application power consumption estimation and feedback for tuning. Cinder complements this work on modeling and accounting. Improved hardware support to determine where energy is going would make its accounting and resource control more accurate. On top of these models, Cinder provides a pair of abstractions that allow applications to flexibly and easily enforce a range of policies.

Energy Efficiency

There is rich prior work on improving the energy efficiency of individual components, such as CPU voltage and frequency scaling [Flautner 2002; Govil 1995], spinning down disks [Douglis 1995; Helmbold 1996], or carefully selecting memory pages [Lebeck 2000]. Phone operating systems today tend to depend on much simpler, but still effective optimization schemes than in the research literature, such as hard timeouts for turning off devices. The exact models or mechanisms used for energy efficiency are orthogonal to Cinder: they allow applications to complete more work within a given power budget. The image viewer described in x5.3 is an example of an energy-adaptive application, as is typical in the Odyssey system [Flinn 1999a].

VII. FUTURE WORK

We believe that the reserve and tap abstractions may be fruitfully applied to other resource allocation problems beyond energy consumption. For instance, the high cost of mobile data plans makes network bits a precious resource. Applications should not be able to run up a user's bill due to expensive data tariffs, just as they should not be able to run down the battery unexpectedly. Since data plans are frequently offered in terms of megabyte quotas, Cinder's mechanisms could be repurposed to limit application network access by replacing the logical battery with a pool of network bytes. Similarly, reserves could also be used to enforce SMS text message quotas. Using the HTC Dream's limited battery level information Cinder could adapt its energy model based on past component and application usage, dynamically refining its costs. Though Cinder can facilitate this, and we have made some adjustments to test this, evaluating the complex and dynamic system this would yield will require additional research.

ISSN: 2321-9653

IX. CONCLUSION

Cinder is an operating system for modern mobile devices. It uses techniques similar to existing systems to model device energy use, while going beyond the capabilities of current operating systems by providing an IPC system that fundamentally accounts for resource usage on behalf of principals. It extends this accounting to add subdivision and delegation, using its reserve and tap abstractions.We have described and applied this system to a variety of applications demonstrating, in particular, their ability to partition applications to energy bounds even with complex policies. Additionally, we showed Cinder facilitates policies which enable efficient use of expensive peripherals despite non-linear power models.

REFERENCES

[1] [Com 1988] THE EXECUTIVE COMPUTER; Compaq Finally Makes a Laptop. http://www.nytimes.com/1988/10/23/ business/the-executive-computer-compaq-finally-makes-a-laptop.html, 1988.

[2] [Fla 2009] Adobe and HTC Bring Flash Platform to Android,June2009.http://www.adobe.com/aboutadobe/press room/pressreleases/pdfs/200906/062409AdobeandHTC.pdf.

[3] [App 2010] Apple Previews iPhone OS 4, April 2010. http://www.apple.com/pr/library/2010/04/08iphoneos.html.

[4] [Economou 2006] Dimitris Economou, Suzanne Rivoire, and Christos Kozyrakis. Full-system power analysis and modeling for server environments. In Proceedings of the 2nd Workshop on Modeling, Benchmarking and Simulation, Boston, MA, 2006.

[5] [Flautner 2002] Krisztian Flautner and Trevor Mudge. Vertigo: automatic performance-setting for linux. In Proceedings of the 5th Symposium on Operating Systems Design and Implementation, pages 105–116, Boston, MA, 2002.

[6] [Flinn 1999a] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In Proceedings of the 17th ACM Symposium on Operating Systems Principles, pages 48– 63, Charleston, SC, 1999.

[7] [Flinn 1999b] Jason Flinn and M. Satyanarayanan. PowerScope: A Tool for Profiling the Energy Usage of Mobile Applications. In Proceedings of the 2nd IEEE Workshop on Mobile Computer Systems and Applications, New Orleans, LA, 1999.

[8] [Govil 1995] Kinshuk Govil, Edwin Chan, and Hal Wasserman. Comparing algorithm for dynamic speed-setting of a low-power CPU. In Proceedings of the 1st Conference on Mobile Computing and Networking, pages 13–25, Berkeley, CA, 1995.

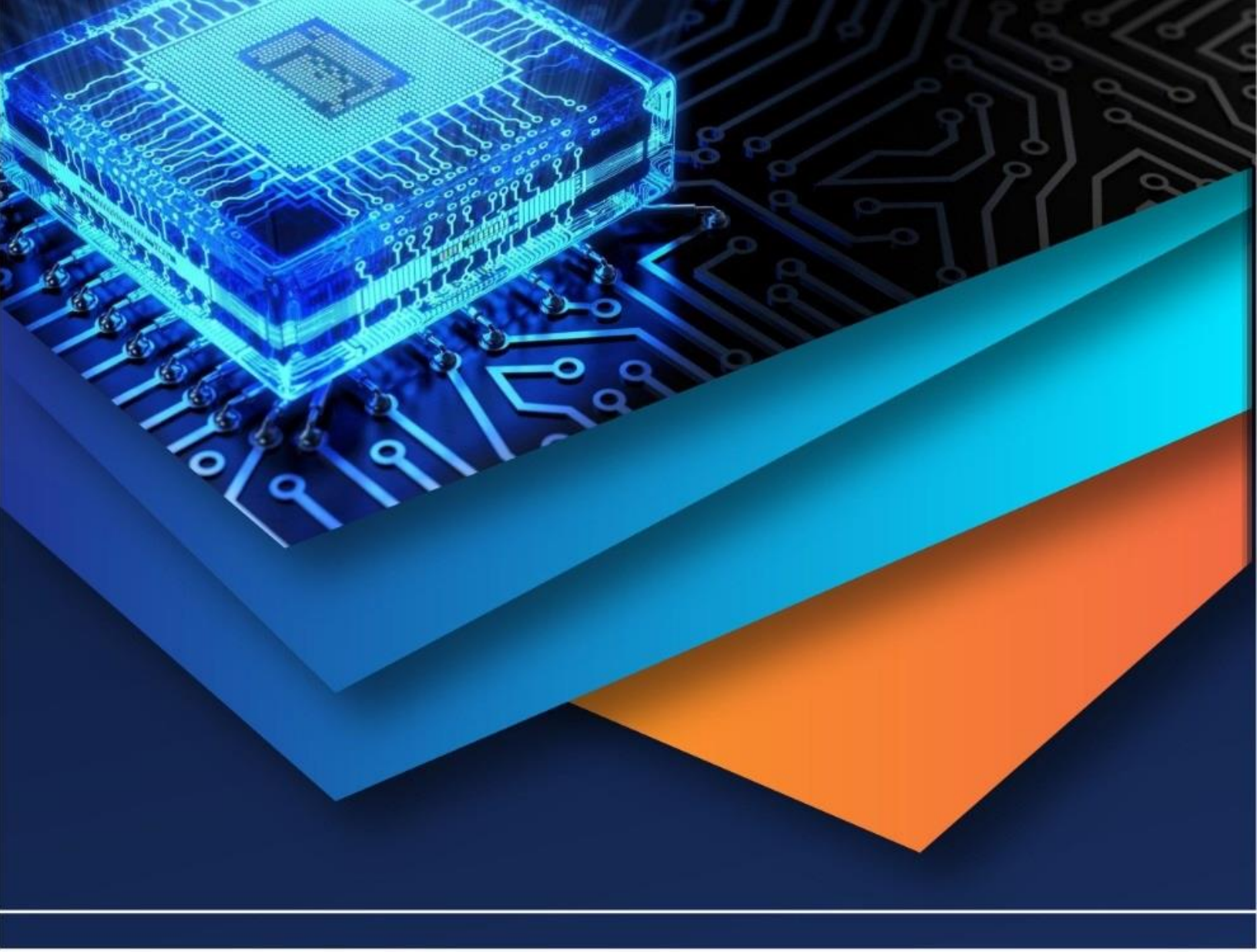
[9] [Helmbold 1996] David P. Helmbold, Darrell D. E. Long, and Bruce Sherrod. A dynamic disk spin down technique for mobile computing. In Proceedings of the 2nd Conference on Mobile Computing and Networking, pages 130–142, Rye, NY, 1996.

[10] [Lebeck 2000] Alvin R. Lebeck, Xiaobo Fan, Heng Zeng, and Carla Ellis. Power aware page allocation. In Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, pages 105– 116, Cambridge, MA, 2000.

[11] [Snowdon 2009] David C. Snowdon, Etienne Le Sueur, Stefan M. Petters, and Gernot Heiser. Koala: a platform for OS-level power management. In Proceedings of the 4th ACM European Conference on Computer Systems, pages 289–302, Nuremberg, Germany, 2009.

[12] [Zeldovich 2006] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazi`eres. Making information flow explicit in HiStar. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation, pages 263–278, Seattle, WA, 2006.

[13] [Zeng 2003] Heng Zeng, Carla S. Ellis, Alvin R. Lebeck, and Amin Vahdat. Currentcy: A unifying abstraction for expressing energy management policies. In Proceedings of the 2003 USENIX Annual Technical Conference, pages 43–56, San Antonio, TX, 2003.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY