www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 1 January 2015, Page No. 9977-9985

A Review on Distributed Application Processing Framework – Clone Cloud

Lect. Prajakta R. Mali[1], Lect. Gayatri D. Naik[2]

[1]Lecturer in Computer Engineering Department
Government Polytechnic, Thane,
pmahajan2013@gmail.com

[2]Lecturer in Computer Engineering Department
YadavraoTasgaonkar Institute of Engineering and Technology
gayatri.naik@tasgaonkartech.com

Abstract: The latest developments in mobile devices technology have made smartphones as the future computing and service access devices. Users expect to run computational intensive applications on Smart Mobile Devices (SMDs) in the same way as powerful stationary computers. However in spite of all the advancements in recent years, SMDs are still low potential computing devices, which are constrained by CPU potentials, memory capacity and battery life time. Mobile Cloud Computing (MCC) is the latest practical solution for alleviating this incapacitation by extending the services and resources of computational clouds to SMDs on demand basis. In MCC, application offloading is ascertained as a software level solution for augmenting application processing capabilities of SMDs. The current offloading algorithms offload computational intensive applications to remote servers by employing different cloud models. This seminar reviews existing Distributed Application Processing Frameworks (DAPFs) for SMDs in MCC domain. And mainly focuses on CloneCloud Framework.

Keywords: Distributed Application Processing Framework, VM based Application Offloading, CloneCloud.

1. Introduction

The miniature nature, compact design, high quality graphics, customized user applications support and multimodal connectivity features have made SMDs a special choice of interest for mobile users. SMDs incorporate the computing potentials of PDAs and voice communication capabilities of ordinary mobile devices by providing support for customized user applications and multimodal connectivity for accessing both cellular and data networks. SMDs employ wireless network technologies for accessing the internet; such as 3G connectivity, Wireless Fidelity (Wi-Fi), Wi-Max, or Long Term Evaluation (LTE). SMDs are the dominant future computing devices with high user expectations for accessing computational intensive applications analogous to powerful stationary computing machines. Examples of such applications include natural language translators, speech recognizers, optical character recognizers, image processors, online games, video processing and wearable devices for patients such as wearable device with a head-up display in the form of eyeglasses (a camera for scene capture and earphones) is a useful application that helps Alzheimer patients in everyday life. Such applications necessitate higher computing power, memory, and battery lifetime on resource constrained SMDs. On the other hand, SMDs are still low potential computing devices having limitations in memory, CPU and battery power. In spite of all the advancements in recent years, SMDs are constrained by weight, size, and intrinsic limitations in wireless medium and mobility.

A key area of mobile computing research focuses on the application layer research for creating new software level solutions. Application offloading is an application layer solution for alleviating resources limitations in SMDs. Successful practices of cloud computing for stationary machines are the motivating factors for leveraging cloud resources and services for SMDs. Cloud computing employs different services provision models for the provision of cloud resources and services to SMDs; such as Software as a Service, Infrastructure as a Service, and Platform as a Service. Several online file storage services are available on cloud server for augmenting storage potentials of client devices; such as Amazon *S*3, Google Docs, MobileMe, and Drop Box. In the same way, Amazon provides cloud computing services in the form of Elastic Cloud Compute. The cloud revolution augments the computing potentials of client devices; such as desktops, laptops, PDAs and smartphones. The aim of MCC is to alleviate resources limitations of SMDs by leveraging computing resources and services of cloud datacenters. MCC is deployed in diverse manners to achieve the aforementioned objective. MCC employs process offloading techniques for augmenting application processing potentials of SMDs.

LITERATURE SERVEY

Muhammad Shiraz, Abdullah Gani, Rashid HafeezKhokhar and RajkumarBuyya [1] review current

DAPFs in SMDs within MCC domain and identify challenges in the cloud based processing of mobile applications. They classify existing DAPFs by thematic taxonomy and investigate commonalities and deviations in such frameworks on the basis of significant parameters such as offloading scope, partitioning approach, migration support, migration granularity, application developer support, migration pattern and execution monitoring. They contribute the categorization of frameworks on the basis of thematic taxonomy, analysis of current DAPFs by discussing the implications and critical aspects, identifying the issues in existing solutions for offload processing and challenge to cloud based application processing of mobile applications. The listing of challenges and open issues guide researchers to select the appropriate domain for future research and obtain ideas for further investigations. They classify the application offloading frameworks by using their attributes as follows-

- **VM Migration Based Application Offloading**
    - Cyber Foraging Framework
    - VM based Cloudlets Framework
    - CloneCloud based framework
    - Elastic Clone cloud framework
    - Mirror server framework
- **Entire Application Migration Based Application Offloading**
    - Universal Mobile Service Cell (UMSC)
    - Distributed Shell System (DISHES)
    - Misco
    - Cogniserve
- **Application Partitioning Based Application Offloading**
    - Static Partitioning Based Application Offloading:
    - Dynamic Partitioning Based Application Offloading:
        - AIDE
        - Mobile Assistance Using Infrastructure (MAUI)
        - Elastic application model

In [4] cyber foraging framework is employed to utilize computation resources of computing devices (stationary or mobile) in close proximity of SMD. The framework implements client/server architecture. Mobile devices request for process offloading and surrogate server provides the services on demand. The frame-work supports configuration of multiple surrogate servers simultaneously and employs virtual machine technology for remote application processing. A single surrogate server is capable to run a configurable number of independent virtual servers with isolation, elasticity, resource control and simple cleanup mechanism. Each offloaded application executes on isolated virtual server.

But R. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb [4] identify some critical aspects of cyber foraging framework is the deployment of template based virtualization approach which is a highly time consuming and resources starving mechanism for VM deployment [41]. The framework requires the annotation of individual components of the application as local or remote which is an additional effort for application developers. Further, surrogate based cyber foraging is restricted to the availability of services and resources on local servers.

A cloudlet [5] architecture proposed by M.Satyanarayan, advocates a two tier approach to decrease the latencies. Proposed architecture states that rather than relying on a distant “Cloud”, we might be able to address the mobile device’s resource poverty via a nearby resource-rich cloudlet. Cloudlets are decentralized and widely dispersed Internet infrastructure components whose compute cycles and storage resources can be leveraged by nearby mobile computers. Access to a cloudlet can be provided by Wi-Fi that saves energy as well as has greater bandwidth as compared to other internet services.

Also MahadevSatyanarayanan†, ParamvirBahl‡, Ramon Caceres•, Nigel Davies [5] identifies the critical aspects of VM based cloudlets framework are that the framework requires additional hardware level support for the implementation of VM technology and is based on cloning mobile device application processing environment to remote host which involves the issues of VM deployment and management on SMD, privacy and access control in migrating the entire execution environment and security threats in the transmission of VM.

2. CloneCloud

In this paper, we take a first step towards realizing this vision, by designing and implementing the first version of the CloneCloud system. CloneCloud boosts unmodified mobile applications by seamlessly off-loading part of their execution from the mobile device onto device clones operating in a computational cloud. It is designed to serve as a platform for generic mobile-device processing as a service. Conceptually, this system automatically transforms a single machine execution (e.g., computation on a smartphone) into a distributed execution that is optimal given the network connection to the cloud, if needed, the relative processing capabilities of the mobile device and cloud, and the application’s computing patterns.
The underlying motivation for such a system lies in the following intuition: as long as execution on the cloud is significantly faster than execution on the mobile device (or more reliable, more secure, etc.), paying the cost for sending the relevant data and code from the device to the cloud and back may be worth it. Only when the metric (e.g., performance or energy) of the newly partitioned application is better than that of the existing application, it makes sense to partition an application. In practice, the partitioning decision may be more fine-grained than a yes/no answer (i.e., it may result in carving off different amounts of the original application for cloud execution). Furthermore, the decision may be impacted not only by the application itself, but also by the expected workload and the execution conditions, such as network connectivity and CPU speeds of both mobile and cloud devices.
A fundamental design goal for CloneCloud is to allow such fine-grained flexibility on what to run where, which traditional client-server partitioning hardwire early on in the development process. Consequently, CloneCloud aims to make application partitioning seamless, and based only on the deployed version of the application, without need for source code. The CloneCloud prototype described here meets all our design goals, by rewriting an unmodified application executable. While the modified executable runs, at automatically chosen points individual threads migrate from the mobile device to a device clone in a cloud. There the thread executes, possibly accessing native features of the hosting platform such as the fast CPU, network, hardware accelerators, storage, etc. Eventually, the thread returns back to the mobile device, along with any state it created abroad, which it merges back into the original process. The choice of where to migrate off and back onto the mobile device is made by a partitioning component,

which uses static analysis to discover constraints on possible migration points, and dynamic profiling to build a cost model for execution and migration.

First, unlike traditional suspend-migrater resume mechanisms for application migration, the CloneCloud migrator operates at thread granularity, an essential consideration for mobile applications, which tend to have features that must remain at the mobile device, such as those accessing the camera or managing the user interface.

Second, unlike past application-layer VM migrators, the CloneCloud migrator allows native system operations to execute both at the mobile device and at its clones in the cloud, harnessing not only raw CPU cloud power, but also system facilities or specialized hardware.

Third, unlike mostly programmer-assisted approaches to application partitioning, the CloneCloud partitioner automatically identifies costs and constraints through static and dynamic code analysis, without the programmer's help, annotations, or application refactoring.

3.1 Augmented Execution

The scope of augmented execution from the infrastructure is fairly broad. In this section, we attempt to categorize the types of augmentation we envision (Figure 3.1).

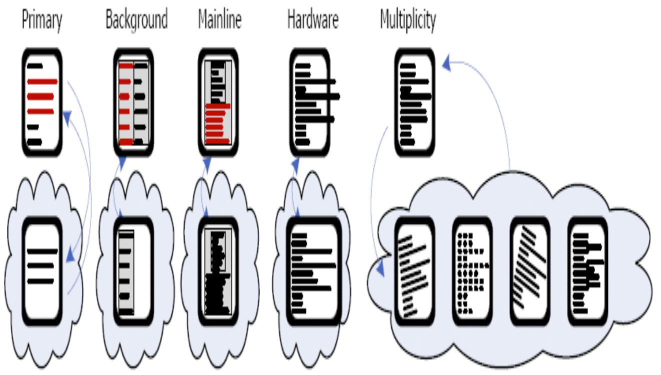

Figure 3.1: The five categories of Augmented Execution

3.1.1 Primary functionality outsourcing:

Computation-hungry applications such as speech processing, video indexing, and super-resolution are automatically split, so that the user-interface and other low-octane processing is retained at the smartphone, while the high-power, expensive computation is off-loaded to the infrastructure, synchronously. This is similar to designing the application as a client-server service, where the infrastructure provides the service (e.g., the translation of speech to text), or as a thin-client environment.

3.1.2 Background augmentation:

Unlike primary functionality outsourcing, this category deals with functionality that does not need to interact with users in a short time scale. Such is functionality that typically happens in the back-ground, such as scanning the file system for viruses, indexing files for faster search, analyzing photos for common faces, crawling news web pages, etc. In this case, entire processes can be marked (by the user or by the programmer) or automatically inferred as "background" processes, and migrated to the infrastructure wholesale. Furthermore, off-loaded functionality can take on the role of a "virtual client." Even when the smartphone is turned off, the virtual client can continue to run background tasks. Later when the smartphone returns online, it can synchronize its state with the infrastructure.

3.1.3 Mainline augmentation:

This category sits between primary functionality outsourcing and background augmentation. Here the user may opt to run a particular application in a wrapped fashion, altering the method of its execution but not its semantics. Examples are private-data leak detection (e.g., to taint-check an application or application set), fault-tolerance (e.g., to employ multi-variant execution analysis to protect the application from trans-parent bugs), or debugging (e.g., keep track dynamically of allocated memory in the heap to catch memory leaks). Unlike background augmentation, mainline augmentation is interspersed in the execution of the application. Many possibilities exist: for example, when a decision point is reached in the taint-check example, the application on the smartphone may block, perhaps causing the clone to rewind back to a known checkpoint, and to re-execute for-ward with taint-tracking, before deciding.

3.1.4 Hardware augmentation:

This category is interestingbecause it compensates for fundamental weaknesses of the smartphone platform, such as memory caps or other constraints, and hardware peculiarities.

3.1.5 Augmentation through multiplicity:

The last category we consider is unique in that it uses multiple copies of the system image executed in different ways. This can help running data parallel applications (e.g., doing indexing for disjoint sets of images). This can also help the application to "see the future," by exhaustively exploring all possible next steps within some small horizon as would be done for model checking or to evaluate in maximum detail all possible choices for a decision before making that decision. Consider, for example, an energy-conserving process scheduler that, in the absence of future knowledge, can only guarantee decisions close but not at the optimum. Instead, the whole system image could be replicated multiple times in the infrastructure, choosing all possible interleaving of processes during execution, and evaluating energy expenditure via some consumption model for the device, ultimately making the scheduling decision that results in the minimum expenditure. In this category of augmentation, infrastructure cycles are lavished on essentially a Monte-Carlo simulation of all possible outcomes of the scheduler's choices to make the optimal decision. We end up wasting much energy (at the infrastructure) to save a little bit of energy on the mobile device. However, given the opportunity cost of being left with a dead battery during a critical time, this rather extravagant use of the infrastructure may have significant benefits.

4.1 Architecture:-

Conceptually, our system provides a way to boost a smart-phone application by utilizing heterogeneous computing platforms through cloning and computation transformation. For doing so, our system (semi)-automatically trans-forms a single-machine execution (e.g., smartphone com-putation) into a distributed execution (e.g., smartphone plus cloud computation) in which the resource-intensive part of the execution is run in powerful clones. An additional benefit of cloning is that if the smartphone is lost or destroyed, the clone can be used as a backup. Figure 2 illustrates the high-level system model of our approach.

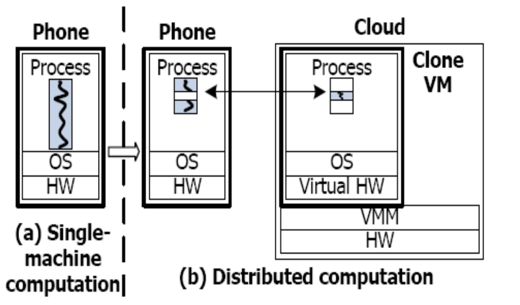

Figure 4.1: CloneCloud system model. CloneCloud transforms a single-machine execution (mobile device computation) into a distributed execution (mobile device and cloud computation) automatically. [3]

Augmented execution is performed in four steps:

1. Initially, a clone of the smartphone is created within the cloud (laptop, desktop, or server nodes);
2. The state of the primary (phone) and the clone is periodically or on-demand synchronized;
3. Application augmentations (whole applications or augmented pieces of applications) are executed in the clone, automatically or upon request; and
4. Results from clone execution are re-integrated back into the smartphone state.

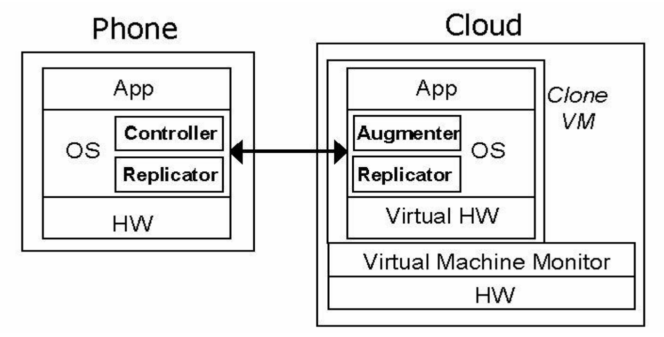

Figure 4.2: Clone execution architecture for smartphones.

The CloneCloud prototype described here meets all our design goals, by rewriting an unmodified application executable. While the modified executable runs, at automatically chosen points individual threads migrate from the mobile device to a device clone in a cloud. There the thread executes, possibly accessing native features of the hosting platform such as the fast CPU, network, hardware accelerators, storage, etc. Eventually, the thread returns back to the mobile device, along with any state it created abroad, which it merges back into the original pro-cess. The choice of where to migrate off and back onto the mobile device is made by a partitioning component, which uses static analysis to discover constraints on possible migration points, and dynamic profiling to build a cost model for execution and migration. A mathematical optimizer chooses migration points that optimize execution time given the application and the cost model.

4.2 Application VMs

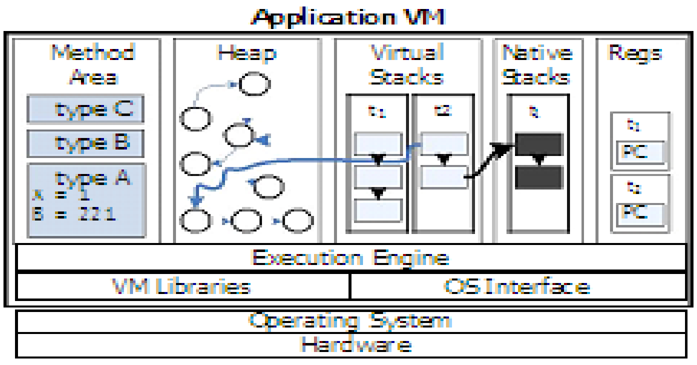

Figure 4.2.1: A general architecture for an application-layer virtual machine.

An application-level VM is an abstract computing ma-chine that provides hardware and operating system in-dependence. Its instruction sets are platform-independent bytecodes; an executable is a blob of byte-codes. The VM runtime executes bytecodes of methods with threads. There is typically a separation between the virtual portion of an execution and the native portion; the former is only expressed in terms of objects directly visible to the bytecode, while the latter include management machinery for the virtual machine itself, data and computation invoked on behalf of a virtual computation, as well the process-level data of the OS process containing the VM. Interfacing between the virtual and the native portion happens via native interface frameworks.

Runtime memory is split between VM-wide and per-thread areas. The Method Area, which contains the types of the executing program and libraries as well as static variable contents, and the Heap, which holds all dynamically allocated data, are VM-wide. Each thread has its own Virtual Stack (stack frames of the virtual hardware), the Virtual Registers (e.g., the program counter), and the Native Stack (containing any native execution frames of a thread, if it has invoked native functions).

Most computation, data structure manipulation, and memory management are done within the abstract ma-chine. However, external processing such as file I/O, networking, using local hardware such as sensors, are done via APIs that punch through the abstract machine into the process's system call interface.

4.3 Partitioning

The partitioning mechanism in CloneCloud aims to modify an application executable by deciding where to execute methods in the code. No special considerations are required for the executable beyond targeting the same application VM; that is, it need not be written in a particular idiom, e.g., a dataflow language. The output of the partitioning mechanism is the executable with partitioning points, optimal for a choice of execution conditions (network link characteristics between mobile device and cloud, relative CPU speeds). The partitioning mechanism can be run multiple times for different execution conditions, resulting in a database that maps partitioning to conditions. At runtime, the distributed execution mechanism we describe in Section 4 implements the choice of partition for the current execution conditions. Partitioning of an application operates according to the conceptual workflow of Figure 4.3.1.

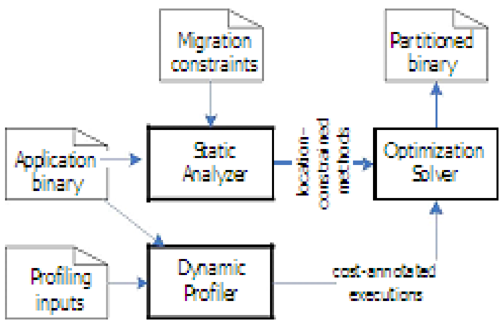

Figure 4.3.1: Partitioning analysis framework.

Our partitioning frame-work combines static program analysis with dynamic program profiling to produce partitioning that optimizes goals while meeting correctness constraints.

The first component, the *Static Analyzer*, identifies le-gal partition choices for the application executable, ac-cording to a set of constraints (Section 3.1). Constraints codify the needs of the distributed execution engine used, as well as the particular usage model we target; however, different mechanisms can seamlessly be plugged into the partitioning component by changing these constraints.

The second component, the *Dynamic Profiler* (Section 3.2), runs the input executable on different platforms (the mobile device and on the cloud clone) under a set of inputs, and returns a set of profiled executions. Profiled executions are used to compose a cost model for the ap-plication under different partitioning.

Finally, the *Optimization Solver* finds a legal partitioning among those enabled by the static analyzer that minimizes an objective function, using the cost model derived by the profiler (Section 3.3). The resulting partitioning is used to modify the executable, yielding the final output of the partitioner. This partitioning is an offline process th t generates a model that the runtime uses.

4.3.1 Static Analyzer

The partitioner uses static analysis to identify legal choices for placing migration and re-integration points in the code. In principle, these points could be placed any-where in the code, but we reduce the available choices to make the optimization problem tractable. In particular, we restrict migration and re-integration points to the entry and exit points, respectively, of methods. In addition, to focus on our application program, we restrict these partitioning points to methods of application classes as op-posed to methods of system classes (e.g., the core classes for Java) or native methods.

4.3.2 Constraints

Three properties required by the migration component of any legal partitioning

Property 1: Methods that access specific features of amachine must be pinned to the machine.

Property 2: Methods that share native state must be collocated at the same machine.

Property 3: Prevent cyclic migration.

4.3.3 Dynamic Profiler

The job of the profiler is to collect the data that will be used to construct a cost model for the application under different execution settings. The cost metric can be different things, including energy expenditure, resource foot-print, etc.; we focus on execution time in the prototype presented here. The profiler is invoked on multiple executions of the application, each using a different set of input data (e.g., command-line arguments and user-interface events), and each executed once on the mobile device and once on the clone in the cloud. The profiler outputs a set S of executions, and for each execution a profile tree T and T ′, from the mobile device and the clone, respectively.

A profile tree is a compact representation of an execution on a single platform. It is a tree with one node for each method invocation in the execution; it is rooted at the starting (user-defined) method invocation of the application (e.g., main). Specific method calls in the execution are represented as edges from the node of the caller method invocation (parent) to the nodes of the callers (children); edge order is not important. Each node is annotated with the cost of its particular invocation in the cost metric (execution time in our case). In addition to its called-method children, every non-leaf node also has a leaf child called its residual node. The residual node I′ for node I represent the residual cost of invocation I that is not due to the calls invoked within I; in other words, node I′ represents the cost of running the body of code excluding the costs of the methods called by it. Finally, each edge is annotated with the state size at the time of invocation of the child node, plus the state size at the end of that invocation; this would be the amount of data that the migrator would need to capture and transmit in both directions, if the edge were to be a migration point. Edges between a node and its residual child have no cost.

4.3.4 Optimization Solver

The purpose of our optimizer is to pick which application methods to migrate to the clone from the mobile device, so as to minimize the expected cost of the partitioned application.

4.4 Distributed Execution

The purpose of the distributed execution mechanism in CloneCloud is to implement a specific partitioning of an application process running inside an application-layer virtual machine, as determined during partitioning.

The lifecycle of a partitioned application is as follows. When the user attempts to launch a partitioned application, current execution conditions (availability of cloud resources and network link characteristics between the mobile device and the cloud) are looked up in a database of pre-computed partitions. The lookup result is a binary, modified with particular migration and re-integrationpoints (special VM instructions in our prototype), which is then launched in a new process. When execution of the process on the mobile device reaches a migration point, the executing thread is suspended and its state (including virtual state, program counter, registers, and stack) is packaged and shipped to a synchronized clone. There, the thread state is instantiated into a new thread with the same stack and reachable heap objects, and then resumed. When the migrated thread reaches a re-integration point, it is similarly suspended and packaged as before, and then shipped back to the mobile device. Finally, the returned packaged thread is merged into the state of the original process. When conditions change, or upon explicit user input via a simple configuration inter-face, a different partition and corresponding binary can be substituted for subsequent invocations of the application.

CloneCloud migration operates at the granularity of a thread. This allows a multi-threaded process to off-load functionality, one thread-at-a-time. CloneCloud enables threads, local and migrated, to use but not migrate native, non-virtualized features of the platform on which they operate: this includes the network, un-virtualized hardware accelerators, natively implemented API functionality (such as expensive-to-virtualize image processing routines), etc. In contrast, most prior work providing application-layer virtual-machine migration keeps native features and functionality exclusively on the original plat-form, only permitting the off-loading of pure, virtualized computation.

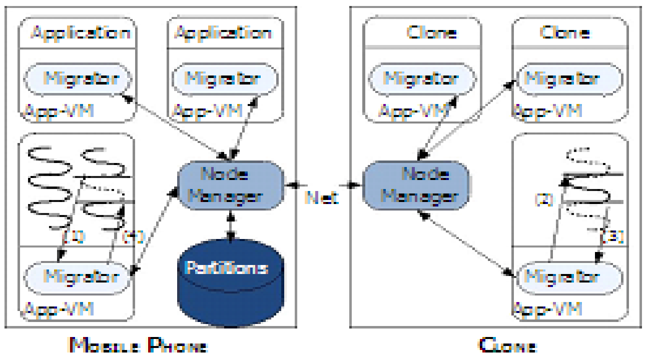

Figure 4.4.1: Migration overview.

These two unique features of CloneCloud, thread-granularity migration and native-everywhere operation, enable new execution models. For example, a mobile application can retain its user interface threads running and interacting with the user, while off-loading worker threads to the cloud if this is beneficial. This would have been impossible with monolithic process or VM suspend-resume migration, since the user would have to migrate to the cloud along with the code. Similarly, a mobile application can migrate a thread that performs heavy 3D rendering operations to a clone with GPUs, without having to modify the original application source; this would have been impossible to do seamlessly if only migration of virtualized computation were allowed.

CloneCloud migration is effected via three distinct components:

(a) A per-process migrator thread that assists a process with the mechanics of suspending, pack-aging, resuming, and merging thread state,

(b) A per-node node manager that handles node-to-node communicationof packaged threads, clone image synchronization and provisioning; and

(c) A simple partition database that determines what partitioning to use.

The migrator functionality manipulates internal state of the application-layer virtual machine; consequently we chose to place it within the same address space as the VM, simplifying the procedure significantly. A manager, in contrast, makes more sense as a per-node component shared by multiple applications, for several reasons. First, it enables application-unspecific node maintenance, including file-system synchronization between the device and the cloud. Second, it amortizes the cost of communicating with the cloud over a single, possibly authenticated and encrypted, transport channel. Finally, it paves the way for future optimizations such as chunk-based or similarity-enhanced data transfer.

4.4.1 Suspend and Capture

Upon reaching a migration point, the job of the thread migrator is to suspend a migrant thread, collect all of its state, and pass that state to the node manager for data transfer. The thread migrator is a native thread, operating within the same address space as the migrant thread, but outside the virtual machine. As such, the migrator has the ability to view and manipulate both native process state and virtualized state.

To capture thread state, the migrator must collect several distinct data sets: execution stack frames and relevant data objects in the process heap, and register con-tents at the migration point. Virtualized stack frames each containing register contents and local object types and contents are readily accessible, since they are maintained by the VM management software. Starting with local data objects in the collected stack frames, the migrator recursively follows references to identify all relevant heap objects, in a manner similar to any mark-and-sweep garbage collector. For each relevant heap object, the migrator stores its field values, and collects all relevant static fields as well (e.g., static class fields).

Captured state must be conditioned for transfer to be portable. First, object field values are stored in network byte order to allow for incompatibilities between differ-ent processor architectures. Second, whereas typically a stack frame contains a local native pointer to the particular class method it executes (which is not portable across ad-dress spaces or processor architectures), we store instead the class name and method name, which are portable.

4.4.2 Resume and Merge

As soon as the captured thread state is transferred to the target clone device, the node manager passes that state to the migrator of a newly allocated process. To resume that migrant thread, the migrator must overlay the thread con-text over the clean process address space. The executable text is loaded (it can be found under the same filename in the synchronized file system of the clone). Then all captured classes and object instances are allocated in the virtual machine's heap, updating static and instance field contents with those from the captured context. As soon as the address space contains all the data relevant to the migrant thread, the thread itself is created, given the stack frames from the capture, the register contents are filled to match the state of the original thread at the migration point in the mobile device, and the thread is marked as runnable to resume execution.

As described above, the cloned thread will eventually reach a reintegration point in its executable, signaling that it should migrate back to the mobile device. Reintegration is almost identical conceptually to the original migration: the clone's migrator captures and packages the thread state, the node manager transfers the capture back to the mobile device, and the migrator in the original process is given the capture for resumption. There is, how-ever, a subtle difference in this reverse migration direction. Whereas in the forward direction from mobile de-vice to clone a captured thread context is used to create a new thread from scratch, in the reverse direction from clone to mobile device the context must update the original thread state to match the changes effected at the clone. We call this process a state merge.

A successful design for merging states in such a fashion depends on our ability to map objects at the original address space to the objects they “became” at the cloned address space; object references themselves are not sufficient in that respect, since in most application-layer VMs, references are implemented as native memory addresses, which look different in different processes, across different devices and possibly architectures, and tend to be reused over time for different objects.

4.4.3 Optimization

The VM offers a unique opportunity for optimizing the amount of information transferred during migration. Be-cause new processes are forked as copies of a "template" process the *Zygote*, in the Android nomenclature—and because that template exists in all booted instances of the Android platform, we can avoid transmitting all sys-tem heap objects that have not changed since an application was copied from Zygote. This typically saves about 40,000 object transmissions with every migration operation, a significant time and bandwidth overhead reduction. Furthermore, even ignoring the transmission cost, some of those objects are static or platform-dependent system objects so should not be migrated anyway.

5. Applications

The uses of Application Migration through CloneCloud are many; as a computing technique it may be applied to many different use scenarios.

- Continuous Speech Recognition
- Augmented Reality
- Image Manipulation
- Video streaming
- Web browsing
- Speech recognition
- 3D rendering
- Selective, application specific fetching of large data sets
- Data mining / Data staging
- Document preparation
- Natural language translation
- Facial recognition
- Text to speech
- Optical character recognition

6. SECURITY ANALYSIS

In cloud computing, offloading of data to the cloud has implications for privacy and security. Because the data is stored and managed in the cloud, security and privacy settings depend on the IT management the cloud provides. A bug or security loophole in the cloud might result in a breach of privacy. For example, in March 2009, a bug in Google caused documents to be shared without the owners' knowledge,9 while a July 2009 breach in Twitter allowed a hacker to obtain confidential documents. Cloud service providers typically work with many third-party vendors, and there is no guarantee as to how these vendors safeguard data. For example, a phishing attack in 2007 duped a staff member for salesforce.com into revealing a password; 13 the attacker then used the password to access confidential data. Obviously, some type of data cannot be stored in the cloud considering the privacy and security issues. One possible solution is to encrypt the data before offloading. But encryption alone cannot solve the problem. A technique called Steganography is also used in the proposed system to hide the data from the cloud vendor.

6.1 Encryption of Data

The data can be encrypted in the mobile system itself before offloading. Here Random Key Generation Algorithm is used. The mobile user can encrypt the data before offloading to the cloud using the random key generated. The cloud vendor before performing computations in it requests for the key to the mobile users, then the cloud vendor after receiving the key decrypts the data and performs computations in it.

6.2 **Steganography**

Steganography is to hide data before sending them to servers so that unauthorized access of data can be prevented. Steganography hides data so that the server is unaware of the existence of information. Image processing is computation-intensive and a good candidate for offloading. Fig.2 shows two examples of Steganography. A coverimage is used to disguise the data image so that the data image is hard to recognize. The combined image is called astego image. A key challenge is to allow offloaded computation to be performed on steganographic data because thecomputation must remain meaningful on stego images. Suppose we want to compare the images in Figure 6.2.1 (b) and (c), Figure 6.2.1 (d) and (e) are sent to the server instead. Figure 6.2.1 (f) shows the pixel-wise difference between (d) and (e). Since the cover image is never sent to server, the server cannot detect hidden data

Fig.6.2.1: Two examples of Steganography. (a) Is the cover image. (b), (c) are hidden in (a) and their corresponding stego images are (d) and (e). (f) is the difference of (d) and (e)

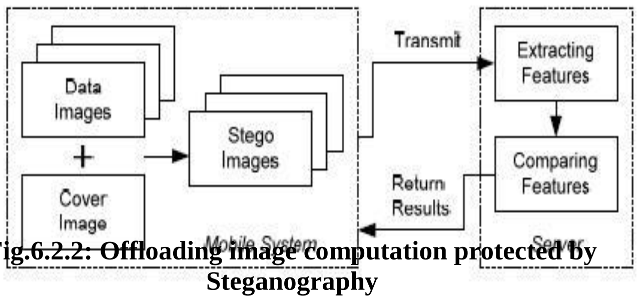

Fig.6.2.2: Offloading image computation protected by Steganography

As shown in Fig.6.2.2, before sending the data to the server, the images are processed using Steganography. The stego images are sent to the server for further processing. The adopted protection techniques must ensure the computation performed at the server remains meaningful. Mean-while, the hidden data must be difficult for the server to detect.

Privacy in the distributed platform and security of data transmission between mobile device and cloud server node are important concerns in cloud based application processing. Privacy measures are required to ensure the execution of mobile application in isolated and trustworthy environment, whereas security procedures are required to protect against network threats. Security and privacy are very important aspects for the establishing and maintaining the trust of mobile users in cloud based application processing. Security in MCC is important from three different perspectives: security for mobile devices, security for data transmission over the wireless medium and security in the cloud datacenter nodes. SMDs are subjected to a number of security threats such as viruses and worms. SMDs are the attractive targets for attacker. According to a report the number of new susceptibilities in mobile operating systems increased 42 percent from 2009 to 2010. The number and sophistication of attacks on mobile phones is increasing speedily as compared to the countermeasures.

Data transmission over the wireless networks is highly vulnerable to network security threats. For example, using radio frequencies, the risk of interruption is higher than with wired networks therefore attacker can easily compromise confidentiality [58]. Similarly, in cloud datacenters the security threats are associated with the transmission between physical elements on the network, and traffic between the virtual elements in the network, such as between virtual machines within a single physical server. Therefore, in order to leverage the application processing services of computational clouds, a highly secure environment is expected at all the three entities of MCC model.

In current DAPFs, transmission of the running states of mobile application which is encapsulated in VM or binary transfer of the application code at runtime is continuously subjected to security threats at mobile device, wireless medium and cloud datacenters. Therefore, secure transmission of the entire components of the application is a challenging issue for MCC. It is imperative to implement reliable security measures for the data transmission, and synchronization between SMD and cloud datacenters in distributed processing platform. Similarly, access control, fidelity and privacy of distributed application components in the remote cloud datacenters is an important consideration for the distributed application processing in MCC.

Cloud datacenters provide augmentation services which are unapproachable to mobile users. Therefore, it is highly demanding to ensure the privacy of data and computing operations in remote server nodes. A trustworthy distributed application model is highly expected to cope with such important issues and ensure the trustworthiness of remote computing environment. A reliable distributed environment is expected to provide authentic access to authorized mobile user for legitimate operations on cloud server nodes.

7. CONCLUSION

This seminar takes a step towards seamlessly interfacing be-tween the mobile and the cloud in the context of mobile cloud computing. Our system overcomes design and implementation challenges to achieve basic augmented execution of mobile applications on the cloud, represent execution of mobile applications on the cloud, representing to address these challenges.

CloneCloud approach is the first to replicate the whole smartphone image and to run the application code with few or no modifications in powerful VM replicas to transform a single-machine computation to a distributed computation (semi)-automatically.

We believe that the CloneCloud architecture enables new, exciting modes of augmented execution for applications in diverse environments, and offers intriguing opportunities for research and for practical deployments that marry the convenience of hand-held devices with the power of cloud computing.

REFERENCES

[1] Muhammad Shiraz, Abdullah Gani, Rashid HafeezKhokhar and RajkumarBuyya," A Review on Distributed Application Processing Frameworks in Smart Mobile devices for Mobile Cloud Computing," in IEEE communications surveys & tutorials, vol. 15, no. 3, third quarter 2013

[2] Byung-Gon Chun, PetrosManiatis,"Augmented Smartphone Applications Through Clone Cloud Execution,"

[3] Byung-Gon Chun†, SunghwanIhm⋆, PetrosManiatis†, MayurNaik†,"CloneCloud: Boosting Mobile Device Applications Through Cloud Clone Execution."in arXiv:1009.3088v2 [cs.DC] 26 Sep 2010

[4] R. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices," in *Proc. 5th international conferenceon Mobile systems, applications and services*. ACM, 2007, pp. 272–285.

[5] Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *Distributed ComputingSystems, Proceedings. 22nd International Conference on*. IEEE, 2002, 217–226.

[6] Y. Su and J. Flinn, "Slingshot: Deploying stateful services in wireless hotspots," in *Proc. 3rd international conference on Mobile systems,applications, and services*. ACM, 2005, pp. 79–92.

[7] J. Porras, O. Riva, and M. Kristensen, "Dynamic resource management and cyber foraging," *Middleware for Network Eccentric and MobileApplications*, vol. 1, p. 349, 2009.

[8] B. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," in *Proc. 8th Workshop on Hot Topics inOperating Systems (HotOS), Monte Verita, Switzerland*, 2009.

[9] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[10] M. Sharifi, S. Kafaie, and O. Kashefi, "A survey and taxonomy of cyber foraging of mobile devices," *IEEE Commun. Surveys Tuts.*, 2011.

[11] X. Zhang, A. Kunjithapatham, S. Jeong, and S. Gibbs, "Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing," *Mobile Networks and Applica-tions*, vol. 16, no. 3, pp. 270–284, 2011.

[12] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computersystems*, vol. 25, no. 6, pp. 599–616, 2009.

[13] K. Kumar and Y. Lu, "Cloud computing for mobile users: Can offloading computation save energy?" *Computer*, vol. 43, no. 4, pp. 51–56, 2010.

[14] "White paper, mobile cloud computing solution brief, aepona," Novem-ber 2010.

[15] H. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *WirelessCommunications and Mobile Computing*, 2011.

[16] S. Abolfazli, Z. Sanaei, and A. Gani, "Mobile cloud computing: A review on smartphone augmentation approaches," in *Proc. 1st Inter-national Conference on Computing, Information Systems and Commu-nications*, 2012.

[17] W. Zheng, P. Xu, X. Huang, and N. Wu, "Design a cloud storage platform for pervasive computing environments," *Cluster Computing*, vol. 13, pp. 141–151, 2010.

[18] M. Satyanarayanan, “Pervasive computing: Vision and challenges,” *IEEE Pers. Commun.*, vol. 8, no. 4, pp. 10–17, 2001.

[19] S. Goyal and J. Carter, “A lightweight secure cyber foraging infrastruc-ture for resource-constrained devices,” in *Mobile Computing SystemsandApplications, 2004. WMCSA 2004. Sixth IEEE Workshop on*. IEEE,2004, pp. 186–195.

[20] C. Li and L. Li, “Energy constrained resource allocation optimization for mobile grids,” *Journal of Parallel and Distributed Computing*, vol. 70, no. 3, pp. 245–258, 2010.

[21] A. Messer, I. Greenberg, P. Bernadat, D. Milojicic, D. Chen, T. Giuli, and X. Gu, “Towards a distributed platform for resource-constrained devices,” in *Distributed Computing Systems, 2002. Proceedings. 22ndInternational Conference on*. IEEE, 2002, pp. 43–51.

[22] A. Dou, V. Kalogeraki, D. Gunopulos, T. Mielikainen, and V. Tuulos, “Misco: a mapreduce framework for mobile systems,” in *Proc. 3rdInternational Conference on PErvasive Technologies Related to Assistive Environments*. ACM, 2010, p. 32.

[23] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “Clonecloud: elastic execution between mobile device and cloud,” in *Proc. sixth conferenceon Computer systems*, 2011, pp. 301–314.

[24] Q. Liu, X. Jian, J. Hu, H. Zhao, and S. Zhang, “An optimized solution for mobile environment using mobile cloud computing,” in *WirelessCommunications, Networking and Mobile Computing, 2009. WiCom’09. 5th International Conference on*. IEEE, 2009, pp. 1–5.

[25] R. Iyer, S. Srinivasan, O. Tickoo, Z. Fang, R. Illikkal, S. Zhang, Chadha, P. Stillwell, and S. Lee, “Cogniserve: Heterogeneous server architecture for large-scale recognition,” *IEEE Micro*, vol. 31, no. 3, pp. 20–31, 2011.

[26] Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, Chandra, and P. Bahl, “Maui: making smartphones last longer with code offload,” in *Proc. 8th international conference on Mobile systems,applications, and services*. ACM, 2010, pp. 49–62.

[27] S. Hung, T. Kuo, C. Shih, J. Shieh, C. Lee, C. Chang, and J. Wei, “A cloud-based virtualized execution environment for mobile applications,” *ZTE Communications*, vol. 9, no. 1, pp. 15–21, 2011.

[28] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107– 113, 2008.

[29] I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso, “Calling the cloud: Enabling mobile phones as interfaces to cloud applications,” *Middleware 2009*, pp. 83–102, 2009.

[30] O. Alliance, “Osgi service platform, core specification, release 4, version 4.1,” *OSGi Specification*, 2007.