JADAVPUR UNIVERSITY

MASTER DEGREE THESIS

A Fault-tolerant Approach to alleviate Faults in Offloading System

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Computer Science and Engineering
of
Jadavpur University

By

Sayanti Mondal
Class Roll No.: 001410502008
Examination Roll No.: M4CSE1607
University Registration No.: 128994 of 2014-15

Under the Guidance of

Dr. Chandreyee Chowdhury
Prof. Sarmistha Neogy

Department of Computer Science and Engineering
Faculty of Engineering and Technology
Jadavpur University, Kolkata
May 2016

Department of Computer Science and Engineering
Faculty of Engineering and Technology
Jadavpur University
Kolkata - 700032, India

C E R T I F I C A T E

This is to certify that the P.G Thesis Work Report entitled **"A Fault-tolerant Approach to alleviate Faults in Offloading System"** submitted by **Sayanti Mondal**(Registration Number: 128994 of 2014-15) as the record of the work carried out by her, is accepted as the P.G Thesis Work Report submitted in partial fulfillment of the requirements for the award of degree of Master of Engineering in Computer Science and Engineering(MCSE) under Department of Computer Science and Engineering, Faculty of Engineering and Technology, Jadavpur University, Kolkata-700032.

....

Dr. Chandreyee Chowdhury
Assistant Professor,
Department of Computer Science
And Engineering,
Jadavpur University,
Kolkata

....

Prof. Sarmistha Neogy
Professor,
Department of Computer
Science and Engineering,
Jadavpur University,
Kolkata

Forwarded By:

....

Prof. Debesh Kumar Das
Head, Department of Computer Science
And Engineering
Jadavpur University,
Kolkata-32

....

Prof. Sivaji Bandyopadhyay
Dean, Faculty of Engineering
and Technology

Department of Computer Science and Engineering
Faculty of Engineering and Technology
Jadavpur University
Kolkata - 700032, India

C E R T I F I C A T E O F A P P R O V A L

The foregoing Thesis is hereby approved as a creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as pre-requisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not endorse or approve any statement made, opinion expressed or conclusion therein but approve this thesis only for the purpose for which it is submitted.

Committee of final Examination
for Evaluation of Thesis

1.______________________________ **2.______________________________**

3.______________________________

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC ETHICS

I hereby declare that this thesis contains literature survey and original research work by the undersigned candidate, as part of her Master of Engineering in Computer Technology studies.

All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name: Sayanti Mondal
Roll Number : 001410502008
Registration Number : 128994 of 2014-15
Thesis Title: A Fault-tolerant Approach to alleviate Faults in Offloading System

Signature with Date: ________________________________

ACKNOWLEDGEMENT

I take this opportunity to express my deepest gratitude and appreciation to all those who have helped me directly or indirectly towards the successful completion of this thesis.

Foremost, I would like to express my sincere gratitude to my respected guide and teacher Dr. Chandreyee Chowdhury (Assistant Professor, Department of Computer Science & Engineering, J.U.) for her exemplary guidance, monitoring and constant encouragement throughout the course of this Project. I gratitude to Dr. Sarmistha Neogy(Associate Professor, Department of Computer Science & Engineering, J.U) and Dr. Sarbani Roy(Associate Professor, Department of Computer Science & Engineering, J.U) for their continuous encouragement, support and guidance. I feel deeply honored that I got the opportunity to work under their guidance.

I would also wish to thank Prof. Sivaji Bandyopadhyay(Dean, FET) and Prof. Debesh Das (Head of the Department of Computer Science & Engineering, J.U) for providing me all the facilities and for their support to the activities of this project.

During the last one year I had the pleasure to work in our department lab. I am grateful to all the members of this lab for their kind co-operation and help.

I would like to express my gratitude and indebtedness to my parents and all my family members for their unbreakable belief, constant encouragement, moral support and guidance.

Last, but not the least, I would like to thank all my classmates of Master of computer science and Engineering(MCSE) batch of 2014-2016, for their co-operation and support. Their wealth of experience has been a source of strength for me throughout the duration of my work.

Abstract

Resources in mobile devices such as battery life, network bandwidth, storage capacity, processor performance are limited. But people are likely to use mobile devices for heavy computations in spite of their resource limitations. To alleviate this problem offloading can be used. Offloading or cyber foraging is an advanced technique to improve the performance of mobile devices by migrating heavy computation to remote powerful servers (such as PDAs, desktop PCs). Many research works have been done so far on the structure of the offloading system, but work on fault tolerance are very few. So this paper mainly concentrates on different faults that occur in offloading system and proposes a fault-tolerance approach to alleviate those faults. First, we have classified all faults in offloading system in four categories- Crash, Omission, Transient and Security. After analyzing the existing fault tolerance approaches, we propose a redundancy based fault-tolerance approach to handle these failures altogether. But our approach is an improved one because it will try to handle all kinds of faults whereas existing approaches, each focus on a specific issue. We have evaluated our fault-tolerance model based on different fault tolerance capability and resource utilization, comparison to other systems such as offloading without fault-tolerance and general execution without offloading.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1: INTORDUCTION

1.1 Offloading Concept

In the last decade people have seen, a wide adoption of advanced mobile devices, called smartphones. These smartphones typically have a rich set of sensors and radios, a relatively powerful mobile processor as well as a substantial amount of internal and external memory. A wide variety of operating systems have been developed to manage these resources, allowing programmers to build custom applications. Centralized market places, like the Apple App Store[1] and the Android Market[2], have eased the publishing of applications. Hence, the number of applications has exploded over the last several years much like the number of webpages did during the early days of the World Wide Web and has resulted in a wide variety of applications, ranging from advanced 3D games [3], to social networking integration applications[4], navigation applications[5], health applications[6] and many more.

So, nowadays mobile devices are not only used for connecting people through phone calls but also for watching videos, playing games, surfing net, biometric authentication, object recognition and so many. With the advancement of technologies, the demand for mobile devices to run heavier applications, increasing every day. Mobile applications are becoming more elaborate in order to support complex applications, such as apps that incorporate augmented reality and high-level mathematical computations. Along with the complicated and rich graphic-intensive characteristic, these applications are still restricted to the hardware resources in the mobile computing environment. Todays' smartphones offer users more applications, more communication bandwidth and more processing, which together put an increasingly heavier burden on its energy usage, while advances in battery capacity do not keep up with the requirements of the modern user. Mobile phones are constrained in their size and weight, their resources like battery life, cpu speed, storage capacity, processor performance are limited.

A very useful technique to alleviate this problem is remote execution: applications can take advantage of the resource-rich infrastructure by migrating the execution to remote servers. As we know that desktop pcs' in offices or cafes remain idle for hours, during this time mobile devices can transfer part of their tasks or the whole task to the resource-rich pc in their vicinity. The pc will then execute the task on behalf of the mobile device. This concept of remote execution through migrating the task to a resourceful device is referred to as offloading or cyber foraging and the server to which remote execution is done known as surrogate. In an offloading system, a surrogate machine makes its resources available to client devices to perform tasks on their behalf. Resource-constrained devices run applications and services that cannot be run on the small devices themselves due to lack of some resource(s). Offloading enables resource-constrained devices to run interesting resource-intensive applications that are beyond their own capabilities. For example, suppose a person wanted to use speech or gesture recognition as inputs to a PDA. But, speech and gesture recognition are compute and power-intensive operations, running far slower than real time on low-power devices [7]. Using cyber foraging, a PDA could capture the voice or gesture data and send it to a powerful surrogate computer to do the recognition. Alternatively, in a future smart home environment, where tiny special-purpose embedded devices can utilize the resources of powerful desktop computers can perform interesting tasks that the devices themselves are incapable of performing. In addition to enabling new applications, cyber foraging can decrease the storage, battery, and computation requirements of embedded or mobile devices, thereby decreasing their size, complexity, and cost. Such devices would need only enough local resources to perform their common tasks, and could use surrogate resources to perform more complex or less common tasks. Fig. 1 shows an offloading scenario.

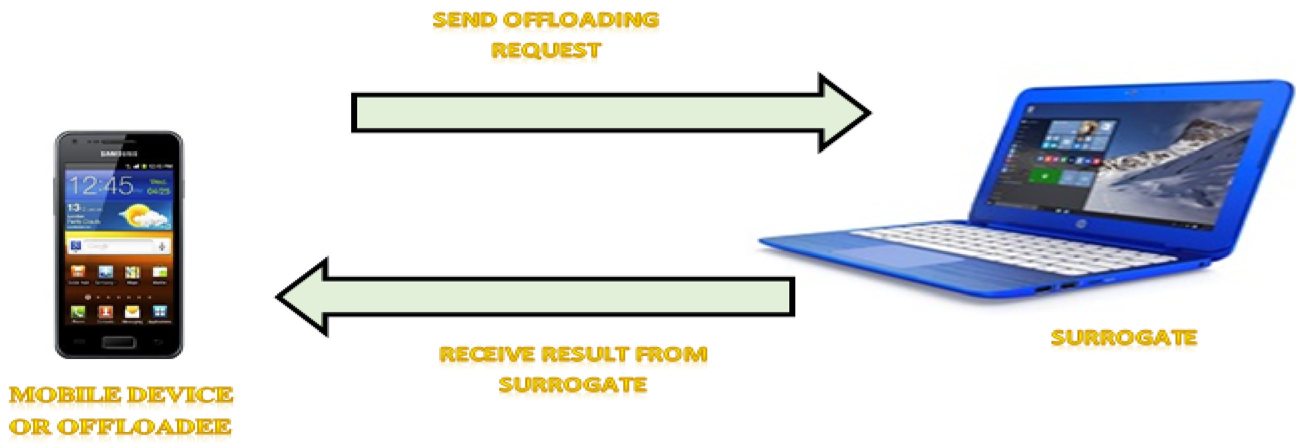


Figure 1: A General Offloading Scenario

1.2 Issues need to be discussed

Offloading is a very popular concept to solve the problem of resource constrained mobile devices. But the drawback is, there are numerous possibilities that will lead to failures in offloading system. If the mobile device and the surrogate are connected via Wi-fi, in that case either connection failure or weak link strength can affect the system. While offloading the surrogate may become unavailable, it may face sudden crash resulting an unsuccessful offloading attempt. All these failures may also lead to data loss. So, fault detection and failure handling schemes are indispensable for offloading system. Huge research works[8-18] have been done so far on the architectural part of this system, some papers[4,5] have also analysed the performance of the offloading system in terms of time, energy consumption. But very few works have been done on analysing different faults in offloading system & how to remove those faults. Most of the papers have considered the offloading system to be fault free, which is unrealistic. The papers which have discussed faults & fault-tolerance techniques have either focused on a single fault or a fault-tolerance method to alleviate specific faults or a single fault. So, we need a system which will analyse all the possible faults that can occur in an offloading system and an overall fault-tolerance mechanism that will tolerate all kind of faults instead of handling some specific faults. So the issues of offloading system that need to be discussed are:

1) Reasons for which offloading may fail.

2) According to their nature classifying them so that all kind of faults can be covered.

3) Finding the existing methods that can handle different faults.

4) Designing a fault-tolerance approach which will tolerate different faults altogether instead of a specific one.

1.3 Objective and Contribution

In our approach, we have tried to find out the reasons for which offloading may fail, like surrogate unreachability, system crash, network failure, mobile nature of the mobile device, security problems etc. After finding out reasons of the failures, according to their nature we have classified them in four failures named- Crash failure, Omission failure, Transient failure and Security failure.

- A system undergoes crash failure when it permanently ceases to execute its actions. This is an irreversible change.
- When a sender sends a sequence of requests to the receiver but the receiver cannot receive one or more requests sent by the sender, or vice-versa then omission failure occurs. In real life, this can be either caused by device malfunction or due to the properties of the medium.
- A transient failure can affect the global state in an arbitrary way. The agent inducing this failure may be momentarily active (like a power surge or a mechanical shock or lightning), but it can make a lasting effect on the global state.
- Virus and other kinds of malicious software creeping into a computer system can lead to unexpected behaviours that lead to security failure. The effects range from allowing an intruder to eavesdrop or steal passwords to granting a complete takeover of the computer system. Such compromised systems can exhibit arbitrary behaviour.

After analysing and classifying the faults, a fault-tolerance approach for offloading system is proposed that will be able to handle different kind of faults altogether. We have used the concept of redundancy to implement fault-tolerance (Details of different fault-tolerance methods are discussed in chapter 3). In this fault-tolerance approach two situations have been considered: Offloading at any location & Offloading at a location where a person visits almost regularly.

When a person visits a location for the first time, her mobile device may either take help of other offloadee(if available) which have visited the location earlier or simply use the redundancy based approach if no offloadee is available nearby and store informations of suitable surrogates of that location. In second step, when the person will visit the place again, offloading decisions will be taken based on those saved information. Redundancy along with prior knowledge has been used here to reduce the occurrence of faults in offloading system. Both the approaches are illustrated in chapter 5. However it is not possible to design a 100% fault-tolerant system, our system will be able to achieve a good percentage towards fault-tolerance.

1.4 Organization of the Thesis

Offloading is an emerging technology; we use exploratory research to make this system more reliable. Thus, our research goal is to identify the factors that affect offloading execution and find a suitable way to alleviate those factors. The structure of this thesis is as follows:
In the next chapter, we discuss about different mechanisms to build offloading system.

Chapter 3 focuses on the reasons for which offloading may fail and categorizing them.

Chapter 4 discusses on the concept of fault-tolerance, and several mechanisms that can be used as fault-tolerant methods for offloading.

In Chapter 5 we present a suitable fault-tolerance mechanism for mobile computation offloading for smartphones.

Next, Chapter 6 describes elaborately about the experimental environment and the benchmark code we have used in our experiment to find the performance of the fault-tolerant offloading system, and finally Chapter 7 concludes this thesis with future work.

Chapter 2: Different mechanisms to build Offloading System

This chapter presents an overview of different offloading systems found in literature.

2.1 MAUI

MAUI[8] decides at runtime which methods should be remotely executed, driven by an optimization engine that achieves the best energy savings possible under the mobile device's current connectivity constrains. It consists of three major components: proxy, profiler and solver. A profiler instruments the program and collects measurements of the program's energy and data transfer requirements. The solver decides, based on input from the MAUI profiler, whether the method in question should be executed locally or remotely. At compile time, MAUI generates two proxies, one that runs on the smartphone and one that runs on the MAUI server, they handle both control and data transfer based on the decision of MAUI solver. Fig. 2 provides an overview of the MAUI architecture.

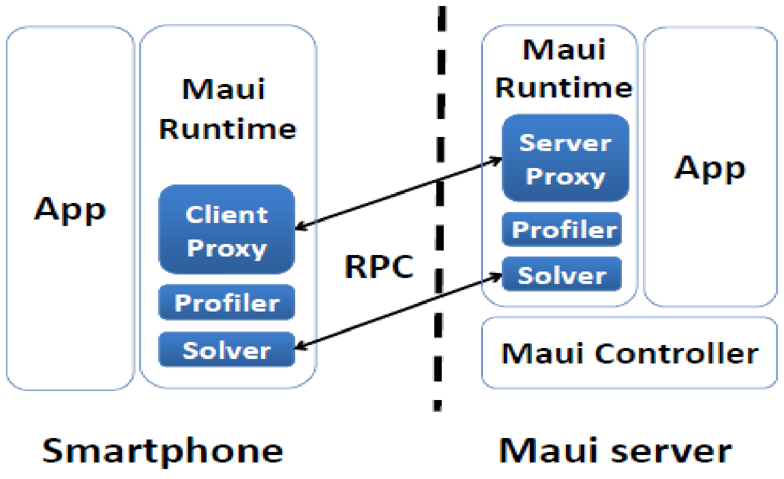


Figure 2: Architecture of MAUI[8]

2.2 Cuckoo

The Cuckoo framework[9], simplifies the development of smartphone applications that benefit from computation offloading and provides a dynamic runtime system, that can, at runtime, decide whether a part of an application will be executed locally or remotely.

Developer creates the project, writes the source code and defines the interface for computer intensive service. Cuckoo system generates a stub/proxy pair for the interface and a remote service with dummy implementation. Developer writes local service implementation, overwrites remote service dummy implementation. System compiles the code and generates an apk file. Then user installs the apk file on its smartphone. In fig. 3 the area within the dashed line shows the extensions by Cuckoo for computation offloading.

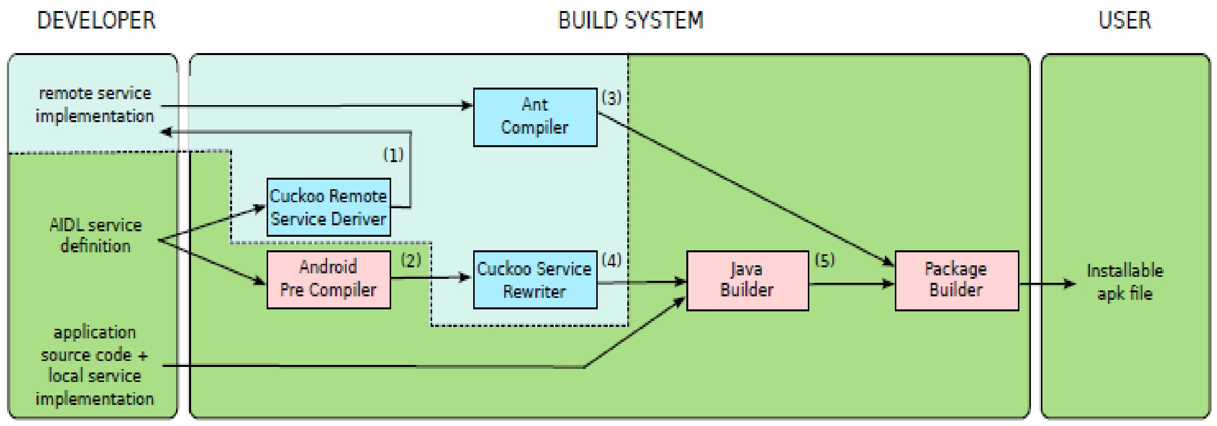


Figure 3: Cuckoo for computation offloading[9]

2.3 Secure cyber foraging

In secure cyber foraging[10], they build their surrogate framework using machine virtualization technology(VServer[19] and Xen [20]).Virtual machine technology allows a single surrogate machine to run a configurable number of independent virtual servers. A PDA user who wants to move a subtask on the surrogate, clicks on the icon associated with an application configured to run, at least partially, on a surrogate machine. If the OS has not already obtained access to a surrogate, it invokes a lightweight surrogate discovery protocol which locates an appropriate surrogate, establishes a service contract with the surrogate for a particular amount of resources, and establishes a security context so that only the client can access the surrogate. The client is given root access to a virtual machine instance. To invoke an application on the surrogate, the client ships a small program to a daemon listening at a known port on the surrogate, which runs the program on behalf of the client. Typically, this program is a shell script that downloads the real application over the Internet, installs it, and runs it. Once the surrogate portion of the application is installed on the surrogate, the application launches the client interface on the device (if any), transparently ships input data to the surrogate portion of the application, collects responses, and outputs them through the client user interface. They use publicly available encryption technologies, both public and private key, as the foundation of security and authentication infrastructure. Fig. 4 shows a simple cyber foraging scenario where a PDA client moves some subtask to a local surrogate machine.

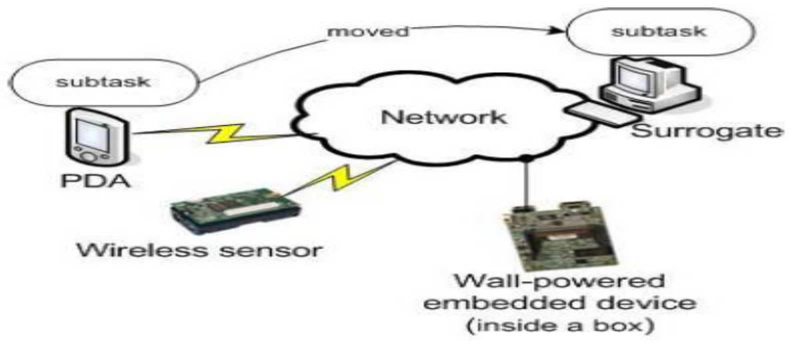


Figure 4: Cyber Foraging scenario[10]

2.4 Offloading Middleware

When a person wants to do offloading using nearby surrogates from a remote server, offloading middleware[11] is required in that situation. Applications running on mobile devices can interact with this middleware directly to invoke its offloading function. Fig. 5 describes an offloading middleware whose main components are:

- Instrumenting module: Used to modify the Java classes (which are in bytecode format) to make them suitable for offloading.
- Partitioning module: Used to partition applications into one local execution partition for running on the mobile device and one or more remote execution partitions for running on surrogates.
- Resource Monitoring module: Monitors the resource usages of the running environment including the mobile device and the underlying network.
- Offloading Decision Engine: Makes partitioning and offloading decisions according to changes in the runtime environment, mainly, the changing in resource utilization.

- Offloading module: Used to execute the partition offloading. Remote execution partitions along with their runtime execution statuses are serialized and migrated to surrogate(s) for remote execution.
- Secure Communication channel: For secure communication between the mobile device and surrogate(s).

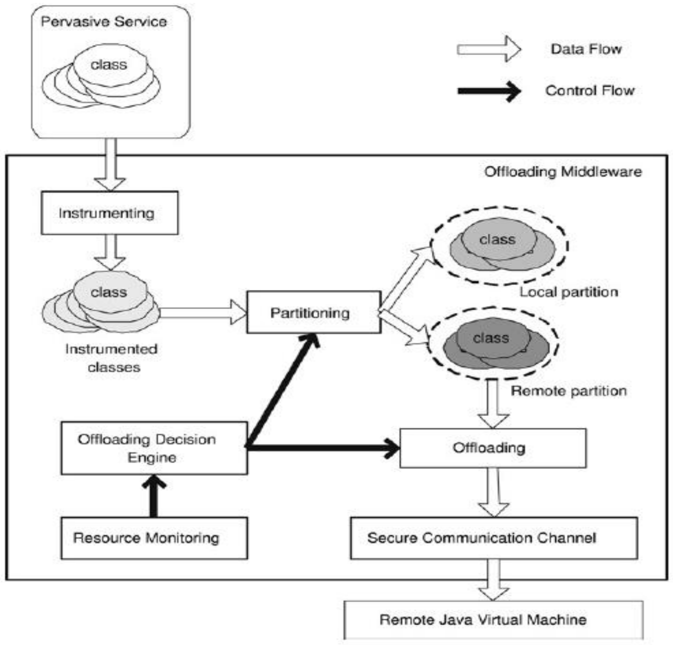


Figure 5: Offloading Middleware[11]

2.5 Scavenger

Scavenger[12] is a cyber foraging system with a new approach towards task distribution and scheduling. Scavenger consists of two independent software components: the daemon running on surrogates enable them to receive and perform tasks, and the library used by client applications. A surrogate installs and run the Scavenger daemon. The daemon consists of a small front-end offering remote access to the mobile code execution environment through some RPC entry points. This front-end is also responsible for device discovery, which it does by using a service discovery framework. Surrogates also collect information about other surrogates, so that this information can be used in future for re-scheduling and task migration, enabling the surrogates to hand over tasks to other surrogates. All client applications use the scavenger library to discover available surrogates. This library offers two ways of working with cyber foraging: 1) a manual mode, where the application may itself ask for a list of available surrogates, install code onto these surrogates, and invoke this code.; and 2) a fully automated mode, where the application programmer only needs to annotate his remote executable functions, and then Scavenger will take care of the scheduling. This paper is developed using the automatic mode. Fig. 6 shows the development of scavenger.

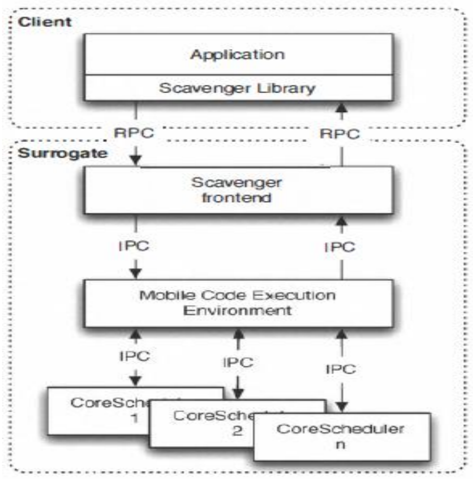


Figure 6: Scavenger[12]

2.6 A runtime offloading approach

This paper[13] proposes a runtime offloading system for pervasive services that considers multiple types of system resources and carries out service partitioning and partition offloading in a more adaptive and efficient manner. The Offloading Toolkit runs on Java Virtual Machine(JVM) consisting of several modules shown in Fig. 7. The offloading toolkit works on java bytecode instead of source code, increasing it's feasibility.

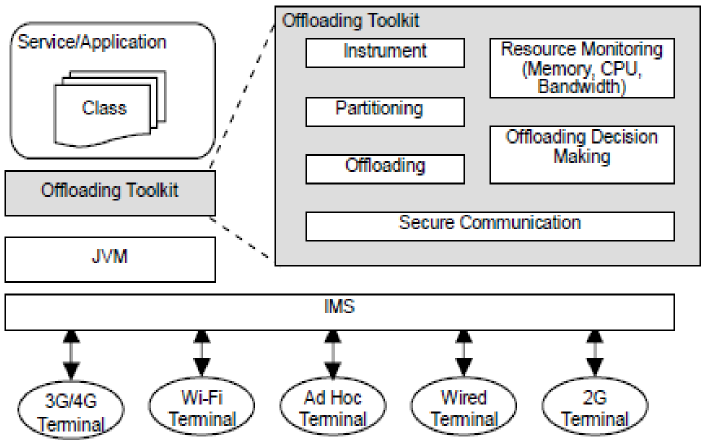


Figure 7: An Offloading toolkit[13]

2.7 Offloading for web-centric devices

This paper[14] proposes a platform-independent mobile offloading system, which is a delegated system for a web centric devices environment. This offloading architecture uses a built-in proxy system that splits the original JavaScript-based application codes into the following two parts: a lightweight code for the mobile client and a computationally heavy code that runs on the server system. This web based offloading architecture requires a built-in proxy, called Contents Adaptor (CA), which basically acts as an intermediary for requests from mobile clients seeking web resources from the outbound servers. A mobile client connects only to the proxy server. The CA analyzes the web resources from the outbound server and determines whether or not the web page needs offloading. If the CA does not find any annotation for the offloading on the code, the web page is transferred to the client with regular proxy server functions, including compression and caching. In contrast, if it detects the offloading annotations in the header of a web page, the CA notifies the generator, which creates a skeleton process for the mobile client on the server side. Concurrently, the CA sends a lightweight modified web page to the client. This offloading method reduces the response time of JavaScript-based web applications and minimizes CPU utilization of mobile devices. Fig. 8. shows the architecture of web-based offloading system.

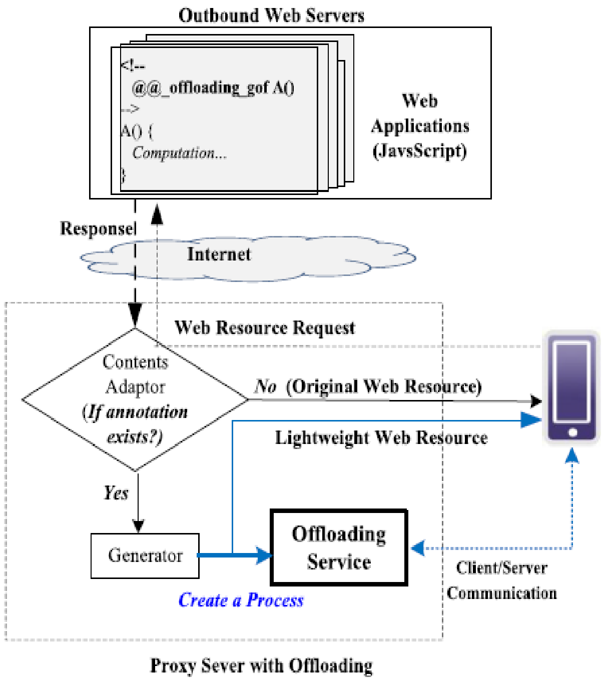


Figure 8: A mobile offloading architecture[14]

2.8 Cloud offloading method for web applications

Concept of cloud offloading has been proposed in paper[16]. They designed the framework for Cloud Offloading for the web applications based on the standard interface named Web Worker in HTML5. With Web Worker[4] web applications can execute the parallelized workload in thread-style. Because of the properties of the Web Worker, the application can do the Cloud Offloading with very little system overhead. The clients such as mobile phones can migrate the web application workload to the servers with the various situations out-of-battery, good network connection. A Web workload balancing Framework(WWF) is shown in Fig. 9.1.

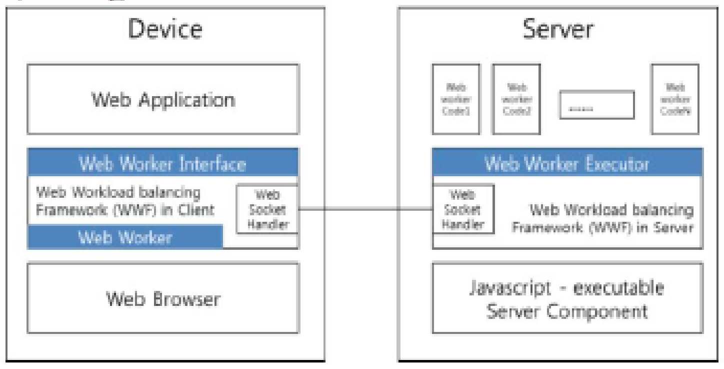

Device
Web Application
Web Worker Interface
Web Workload balancing Framework (WWF) in Client
Web Socket Handler
Web Worker
Web Browser
Server
Web Worker Executor
Web Socket Handler
Web Workload balancing Framework (WWF) in Server
Javascript - executable Server Component

Figure 9.1: Web workload balancing framework for cloud offloading[16]

In the device side, WWF provides the wrapping interface the same as Web   Worker to provide the same interface to the web applications. The device-side WWF will connect the server-side WWF via Web Socket interface to delivery messages between the Web Worker and the original web application. In the server-side WWF, the JavaScript executable server component will execute the web worker code in the server and it will communicate the device-side WWF via Web Socket. When the web application opens and set to use Cloud Offloading, one Web Socket opens between the device-side WWF and the server-side WWF and all commands from device-side and all events in server-side will be delivered via WWF. Node.js has been used as the server here because node.js is the scripting engine for JavaScript language.

2.9 Distributed dynamic offloading system

A fine-grained runtime offloading system, called adaptive infrastructure for distributed execution (AIDE), has been proposed in paper[17]. The key idea is to dynamically partition the application during runtime, and migrate part of the application execution to a powerful nearby surrogate device. In this paper, an offloading inference engine (OLIE) is presented, which makes intelligent offloading decisions to enable AIDE to deliver applications on resource-constrained mobile devices with minimum overhead. Two important decision-making problems solved by OLIE: (1)timely triggering of adaptive offloading, and (2) intelligent selection of an application partitioning policy. To solve the first problem, OLIE decides when to trigger the offloading action. If an offloading action is triggered, OLIE decides the new level of memory utilization to employ on the mobile device given the current resource conditions (e.g., wireless network bandwidth) in the pervasive computing environment. To solve the second problem, OLIE selects a proper application partitioning policy that decides which program objects should be offloaded to the surrogate and which program objects should be pulled back to the mobile device during an offloading action. To achieve both flexibility and stability, OLIE employs the Fuzzy Control model for making offloading decisions. Architecture of runtime offloading system shown in fig. 10.

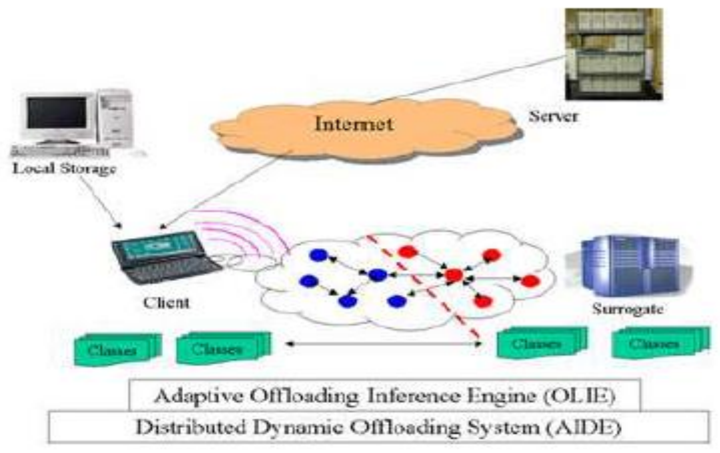

Internet
Server
Local Storage
Client
Surrogate
Classes
Classes
Classes
Classes
Adaptive Offloading Inference Engine (OLIE)
Distributed Dynamic Offloading System (AIDE)

Figure 10: Distributed dynamic offloading system architecture[17]

2.10 Clonecloud

Paper[18] presents the design and implementation of CloneCloud, a system that automatically transforms mobile applications to benefit from the cloud. The system is a flexible application partitioner and execution runtime that enables unmodified mobile applications running in an application-level virtual machine to seamlessly offload part of their execution from mobile devices onto device clones operating in a computational cloud. CloneCloud(Fig. 11) uses a combination of static analysis and dynamic profiling to partition applications automatically at a fine granularity while optimizing execution time and energy use for a target computation and communication environment. At runtime, the application partitioning is effected by migrating a thread from the mobile device at a chosen point to the clone in the cloud, executing there for the remainder of the partition, and re-integrating the migrated thread back to the mobile device. Their evaluation shows that CloneCloud can adapt application partitioning to different environments, and can help some applications achieve as much as a 20x execution speed-up and a 20-fold decrease of energy spent on the mobile device. Since application-layer VMs are widely used in mobile platforms, work in this paper applies primarily to application layer virtual machines (VMs), such as the Java VM, Dalvik VM from the Android Platform, and Microsoft's .NET.

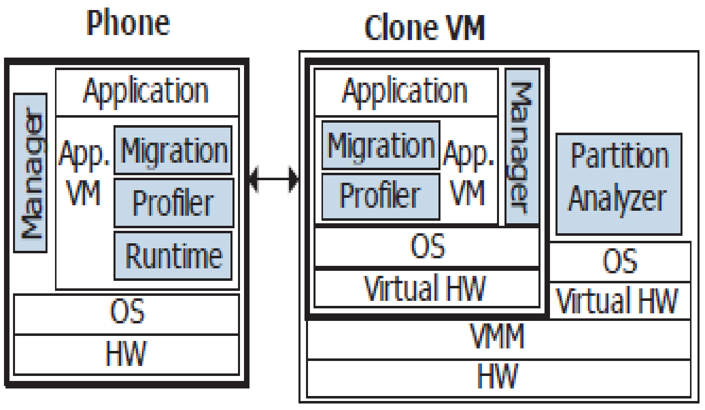


Figure 11: CloneCloud prototype architecture[18]

Summary

Offloading system can be built in different ways for different purposes. In some systems offloading decision is predefined whreas in others the decision is taken during runtime. Sometimes offloading systems are built for web-centric application, some are built for resorce intensive computations. So, different applications implement different approaches of offloading framework. But every sysem is vulnerable to faults. Next chapter discusses different faults in offloading system.

Chapter 3: Fault Detection and Classification in Offloading System

This chapter first describes general faults that occur in a distributed system, then dicusses the faults in offloading system and classifies them in general faults.

3.1 Concept of Fault

Fault is the manifestation of an unexpected behavior of a system[22]. When a system behaves erroneously/differently deviating from its normal execution we can say that ststem is facing a fault. Since distributed systems consist of large number of geographically seperated components to perform critical as well as noncritical tasks, they are very prone to failures. Bad system designs and behavioral patterns like mobility contribute to failures.

Fault, error and faillure- these terms are closely interrelated. The widely accepted definition, given by Avizienis and Laprie[26] is as follows. A fault is a violation of a system's underlying assumptions. An error is an internal state that reflects a fault. A failure is an externally visible deviation from specifications. A fault need not result in an error, nor an error in a failure. An alpha particle corrupting a memory location is a fault. If that memory location contains data, that corrupted data is an error. If a program crashes because of using that data, it is a failure. So, basically fault is the root cause, error is the result of fault and failure is the final outcome.

3.2 Classification of Failures

3.2.1 Crash failure

When a system undergoes crash failure it stops execution. This may be permanent or temporary. When the effect of crash failure is temporary i.e. a system recovers from the failure after a finite period of time, then such a crash failure is known as napping failure.

Operating system failure is one example of crash failure.

In an asynchronous model, crash failures cannot be detected with total certainty, since there is no lower bound of the speed at which a process can execute. But in a synchronous system processor speed and channel delays are bounded, so a crash failure can be detected using timeout.

3.2.2 Fail-stop failure

In this type of failure, the server only exhibits crash failures, but at the same time, we can assume that any correct server in the system can detect that this particular server has failed.
A fail-stop processor has two properties- (1) it halts program execution when a failure occurs and (2) the internal state of the volatile storage is irreversibly lost. A k-fail-stop processor satisfies fail-stop properties with high probability when k or fewer faults occur, and the system can detect when another fail-stop processor halts. If a system cannot tolerate fail-stop failures then there is no way it can tolerate crash failure.

3.2.3 Omission failure

Suppose a transmitter process sending a sequence of messages to a receiver process. If the receiver does not receive one or more of the messages sent by the transmitter, then an omission failure occurs. In real life, this can be either caused by transmitter malfunction or due to the properties of the medium. For example, limited buffer capacity in the routers can cause some communication systems to drop packets. In wireless communication, messages are lost when collisions occur in the MAC layer or the receiving nodes moves out of range.

3.2.4 Transient failure

A transient failure can perturb the global state in an arbitrary way. The agent inducing this failure may be momentarily active(Like a power surge, or a mechanical shock, or lightning), but it can make a lasting effect on the global state. Transient failures are also caused by an overloaded power supply or due to week batteries.

3.2.5 Byzantine failure

Byzantine failure allow every conceivable form of erroneous behaviour. Byzantine failures are known as arbitrary failures and these failures are caused across the distributed systems. These failures cause the system to behave arbitrary in nature. Output from the system would be inappropriate and there could be chances of the malicious events and duplicate messages from the server side and the clients get arbitrary and unwanted duplicate updates from the server due to these failures.
Possible causes of byzantine failure are:
1) Total or partial breakdown of the link connecting the client and server
2) Software problems in a process
3) Due to malicious action in any part of the system

3.2.6 Software failure

There are several reasons that lead to software failure:

1) Coding errors or human errors- Due to erroneous coding, the system may enter into an infinite loop or may result other internal errors.
2) Software design errors
3) Memory leaks- The execution of programs suffer from the degeneration of the running system due to memory leaks, leading to system crash. Memory leak is a phenomenon by which processes fail to free up the entire physical memory that has been allocated to them.
4) Problem with inadequacy of specification- If a system suddenly fails to produce the intended results even if there is no hardware failure or memory leak, then there may be a problem with specifications.

3.2.7 Temporal failure

Real-time systems require actions to be completed within a specific amount of time. When this deadline is not met, a temporal failure occurs. Like software failures, temporal failures also can lead to other types of faulty behaviors.

3.2.8 Security failure

Virus and other kinds of malicious software creeping into a computer system can lead to unexpected behaviors that conform to the definition of a fault. The effects range from allowing an intruder to eavesdrop or steal passwords to granting a complete takeover of the computer system. Such compromised system can exhibit arbitrary behavior.

3.3 Faults in Offloading System

Since offloading system is based on client-server model, it is susceptible to several faults. The mobile nature of mobile devices, the unstable connectivity of wireless links all render a less predictability of the performance of an application running under the control of offloading systems. Several faults that occur in offloading system are classified into four categories: Crash, Omission, Transient and Security[22].

Crash failure: During offloading sometimes it happens that, either the surrogate or the mobile device stops working. The surrogate stops working due to sudden crash, or automatically shuts down due to power cut. Whereas the mobile device stops working when it hangs for multitasking overloads or switches off automatically due to running out of battery. These situations lead to crash failure in offloading system resulting an unexpected halt.

• One reason behind crash failure in offloading system is Software aging[23]. Software aging is a common phenomenon in computer systems which cause frequent software faults, resulting in system outage. Software aging refers, accumulation of errors during continuous operation of the software for a long period of time. In software aging, performance start degrading gradually and finally results in hang/crash failure. Some typical causes of aging are exhaustion of operating system resources, data corruption, storage space unavailability and accumulation of numerical round-off errors.

Omission failure: During offloading, the request sent by the mobile device may not reach the surrogate or the result sent by the surrogate may not reach the mobile. Either the request or the result may be lost during transmission. These kind of failures in offloading system can be classified as omission failures.

• Omission failure in offloading system generally occurs for unreliable network[23,24]. Low bandwidth, long delays are possible causes incurring network unreliability. While migrating the computation to the surrogate, the execution of the offloading task may suffer from delays or even failures by unreliable network.

• Another reason for omission failure is unavailability of the surrogates[25]. The mobile nature of the mobile device in the offloading system affects the availability of the surrogates. Naturally, if the surrogate is not available within the range of the mobile device then omission failure occurs.

Transient failure: Sometimes offloading system is affected by unknown, hidden agents. These agents affect the system in an arbitrary way and perturb the system functionality temporarily. These kind of faults occur for a very short duration of time making them very hard to detect. For example, an offloading request sent by the mobile device not reached the surrogate even if there is no network failure.

• Software bugs or failures lead to transient failure[27] in offloading system. There are faults in a software that have escaped all possible analysis & testing. They are likely to manifest in protracted executions of applications, eventually interfering with their longevity. These transient failures are unpredictable & lead to costly after effects. They might corrupt a database far beyond repair without leaving a trail, they might cause memory leakage which will eventually crash the process or they might induce slow chocking of other operating system resources, eventually paralyzing an entire application.

Security failure: Along with the benefits of high performance, offloading system witnesses potential security threats including compromised data due to the increased number of devices, parties & applications involved, which increases the number of point of access. Offloading system is susceptible to security threats like snooping, spoofing and 'man in the middle attack'. When the mobile device & the surrogate is communicating with each other, a person or a device sitting in between can eavesdrop and gain access to all the information flowing between them, which is known as snooping. But when the person use those information for hacking or deception then it is known as spoofing. 'Man in the middle attack' occurs when the attacker secretly intercepts and relays messages between the mobile device and the surrogate but both of them believe they are communicating directly with each other.

• Timing attack[28,29] causes specific security failures in offloading system. It enables an attacker to extract secrets maintained in a system by observing the time it takes to respond to various queries.

1.Mobile offloading requires access to resourceful servers for short duration through wireless networks. The servers may use virtualization techniques to provide services so that they can isolate & protect different programs & their data. In timing attack the attacker can bypass the isolated environment provided by virtualization characteristics, where sensitive code is executed in isolation from untrustworthy applications.

2.In the offloading system, we consider a server master key is used for encryption & decryption operations of user data. In timing attack to offloading system, an attacker will continue to send requests to the server & the obtained service will be properly performed by the server. The attacker records each response time for a certain service & tries to find clues to the master secret of the server by comparing time differences from several request queues. If the attacker successfully breaks the secret information from the timing results, he can read & even modify other users' information without authorization.

Chapter 4: Existing Fault-tolerance Approaches For Offloading

This chapter discusses on the concept of fault-tolerance, different fault-tolerant systems and focuses on some existing mechanisms that can be used to tolerate faults in offloading system.

4.1 Concept of fault-tolerance

Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of (or one or more faults within) some of its components. A fault-tolerant design enables a system to continue its intended operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. That is, the system as a whole is not stopped due to problems either in the hardware or the software.

Fault-tolerance is needed in order to provide three main features: Reliability, Availability and Security. Reliability focuses on a continuous service without any interruptions, Availability is concerned with readiness of the system and Security prevents any unauthorized access.

Examples- Patient monitoring systems, flight control systems, Banking services.

4.2 Fault-Tolerant Systems

A system is called fault-tolerant when the system maintains or returns to its original configuration after all faulty actions stop executing. There are four major types of fault-tolerance:

4.2.1 Masking Tolerance

A fault is masked if its occurrence has no impact on the application. Masking tolerance is very important in many safety-critical applications where failure can endanger human life or cause massive loss of property. For example- A patient monitoring system in a hospital must not record patient data incorrectly even if some of the sensors or instruments malfunction, since this can potentially cause an improper dose of medicine to be administered to the patient and endanger his/her life.

4.2.2 Nonmasking Tolerance

In nonmasking fault-tolerance, faults may temporarily affect the application but eventually normal behavior is restored. For example-While watching movies even if the server crashed, but the system automatically restored the service by switching to a standby server.

4.2.3 Fail-Safe Tolerance

Certain faulty configurations do not affect the application in an adverse way and therefore considered harmless. A fail-safe system relaxes the tolerance requirement by only avoiding those faulty configurations that may have catastrophic consequences, even when failures occur. For example- if at a four way traffic crossing, the lights are green in both directions, then a collision is possible. But if the lights are red in both directions then at best traffic will stall but will not have any catastrophic side effect.

4.2.4 Graceful Degradation

There are systems that neither mask nor fully recover from the effect of failures, such systems exhibit a degraded behavior that falls short of the normal behavior, but are still considered acceptable. For example- While routing a message between two points in a network, a program computes the shortest path. In the presence of failure, if this program returns another path that is not the shortest but one that is marginally longer than the shortest one, then this may considered acceptable.

4.3 Fault-tolerance mechanisms for offloading

4.3.1 Checkpointing

Paper[9] considers checkpointing to achieve fault-tolerance in offloading system. The system periodically makes checkpoints by taking snapshots of the application execution in both MH and Surrogate side at a particular interval. During failure recovery operation, the offloading system loads the latest checkpoint data and recovers the application execution to the point of the latest checkpointing. This approach may handle failures that occur in the mobile device due to running out of battery, abnormal shutdown and also in the surrogate side owing to surrogate failures such as shutdown or wireless link failures or the surrogate becoming unreachable due to MH’s movement.

4.3.2 Timeout Mechanism

MAUI[8] detects failures using a simple timeout mechanism: when the smartphone loses contact with the server and the server executing a remote method, MAUI returns the control back to the local proxy. At this point, the proxy can either re-invoke the method locally, or it can attempt to find an alternate MAUI server to reinvoke the method on that new server. This approach can handle the omission failure.

4.3.3 Software Rejuvenation

Software Rejuvenation[30,31] is proposed to counteract aging in paper[23]. In software rejuvenation, running software is occasionally stopped, then the accured errors are removed & the software is restarted. Software rejuvenation is the concept of periodically & preemptively restarting an application at a clean internal state after every rejuvenation interval. During rejuvenation the system performance temporarily degrades but the server stability is guaranteed by avoiding software aging.

4.3.4 Restart Mechanism

Paper[23] indicates that the poor network quality is the primary cause of performance deterioration in offloading system and theoretically proved restart[24] can improve this situation. Paper[33] has also considered restart as a simple recovery scheme to mitigate network failures. Markov chain models and Laplace transforms have been developed to analyze the performance of restart for improving the expected task completion time [34,35]. These analyses strongly support the efficiency of restart if the best restart timeouts known. For a given random variable T describing task completion time, restart after a timeout τ is promising if the following condition holds:

$$E[T] < E[T - \tau | T > \tau] \qquad (1)$$

The condition interprets that for restart to be beneficial the expected completion time when restarting from scratch must be less than the expected time still needed to wait for completion. It can be shown [33] that condition (1) holds if the task completion time follows a distribution with sufficiently high variance or heavy-tail.

4.3.5 Homomorphic Encryption

Paper [36] uses homomorphic encryption to implement security in computation offloading. Homomorphic encryption allows computation to be performed on encrypted data without decryption, preserving security and privacy to the offloaded data at server side. This paper adopts homomorphic encryption to protect data in image retrieval technique. In this technique, images are first encrypted then sent to server. The server never decrypts the data, it can only access the encrypted data. Thus the attacker can only launch ciphertext-only attacks and they are the weakest form of attacks. This attacks can be further weakened by changing the keys during encryption.

4.3.6 Steganography

In paper [37] steganography is used to retrieve similar images. Image retrieval finds images similar to a query by extracting images' features and comparing the features. It is computation intensive and thus a good candidate for computation offloading. To protect privacy from unauthorized use by the server in image retrieval technique, the concept of steganography is used here. In this technique, a cover image is used to disguise the data image so that the data image is hard to recognize. The combined image is called a stego image which is sent to the server. Server cannot easily detect hidden data if the server does not know what steganographic technique is adopted.

4.3.7 Protocol based method

In paper[14] several mechanisms have been implemented to handle the situation when the network between the mobile device and the offloading server is not available or overloaded. If the client does not get response from the offloading server in a timely manner, then it sends duplicate request to the server. In this offloading system each request has its own sequence number, so that a duplicate request can be detected. If a proxy server detects a duplicate page request within a certain time period, it understands the failure in the communication channel. The system immediately switches to non-offloading method, since it assumes that the offloading system may have encountered a critical problem. But if the network failure happens while the application is running then the application need to be reloaded.

4.3.8 Using Surrogate discovery mechanisms

Due to mobile nature in mobile host surrogate reachability becomes a problem in offloading system. One way of handling mobility is tracking surrogate reachability employing surrogate discovery mechanisms. In a mobile wireless environment, the location of the surrogates can be obtained by employing surrogate discovery mechanisms, such as CROSS[38]. CROSS is a combined routing & surrogate selection algorithm for pervasive service offloading in mobile ad hoc environment. In this algorithm, a surrogate requirement message (SREQ) is used to specify minimum requirement of available resources in the surrogate. During surrogate discovery, procedures need to find out all the surrogates in the mobile ad hoc environment who satisfy SREQ. Now, surrogate discovery can be incorporated into the existing distributed routing protocols. For the proactive protocols, such as OLSR[39] the capability information of a surrogate can be piggybacked in the link state advertisements(LSAs). Along with the dissemination of the LSAs, the mobile host knows the resource availability of all the surrogates in the environment. For reactive routing protocols, such as DSR[40], the SREQ can be piggybacked in the route request(RREQ)message. Only the surrogates fulfilling the SREQ can send back a route reply(RREP) message. Therefore, no matters which kind of routing protocol utilized in the mobile ad hoc environment, the mobile host obtains a partial view of the network which includes a list of surrogates satisfying SREQ & the paths to these surrogates. Now, given surrogates' locations the surrogate unreachability α can be obtained. If pr{SR} represents surrogate reachability then $pr\{SR\} = 1 - \alpha$

4.3.9 Local re-execution

Connection failure occurs due to two reasons-unreliable network and cloud servers experiencing a long downtime. These situations cannot be avoided completely during the execution of offloading, so failure schemes are indispensable. In paper[33], local re-execution has been considered as a failure handling scheme in offloading system. Usually, there are two major schemes to solve the problem of connection failure. The first one is halting the current execution state and waiting for the network recovery. After the connection quality satisfies the offloading condition again, the execution of offloading is resumed. If the wireless network recovers quickly back to the required level, this scheme performs well. But when the network is failure prone this scheme is not suitable, long time will be wasted accompanied with large energy consumption. In this case, local re-execution may lead to better option. But local re-execution is most effective when it is launched after an optimal time waiting for network recovery. Because, if the wireless network recovers within a moderate time, resuming the execution of offloading may still require less time and energy than the local re-execution. Therefore, it is worth to wait such a period for the connection recovery. This paper analyzes the performance of locally re-executing the offloaded tasks for handling connection failure in mobile offloading, and proposes a method to find the appropriate moment for launching the local re-execution.

4.3.10 Rekeying Mechanism

Timing attack is one of the security issues in offloading system which cannot be prevented by traditional cryptographic security mechanisms. Paper[24] has discussed on this issue which can be solved by rekeying i.e. frequently changing the key[40]. In the offloading systems they consider, a server master key is used for the encryption and decryption operations of user data. To improve security, server should regularly change the master key. But for that the mobile host need to use both new & old master key to access the server(surrogate). When user data is very large this process will take long. Therefore, it is required to recommend a minimum time for the master key replacement cycle & select a suitable time, when there is low amount of user access (e.g. at night).

Summary

Above discussed mechanisms can tolerate specific faults in offloading system, so we need a mechanism that is capable of handling all kind of faults. Chapter 5 proposes a redundancy based fault-tolerance approach that can tolerate different type of faults in offloading system altogether.

Chapter 5: Redundancy based Fault-tolerance Approach for Offloading

This chapter proposes a redundancy based fault-tolerance approach to alleviate faults in offloading system. This approach also uses the concept of timeout mechanism and location based dependency.

5.1 Introduction

In general, two commonly used fault-tolerant techniques are- Checkpointing and Redundancy. Every system has some information associated with it which defines its state at a particular moment. This information includes process state, environment, value of active registers and variable. All this information are collected and stored and each such instance is called a checkpoint. In the event of a failure, system is restored to a previously stored checkpoint rather than starting it from beginning whereas, redundancy is creating multiple copies or replicas of data items and storing them at different locations. This will increase the availability, so that if one node fails still data can be accessed from other node.

5.2 Our Proposal

Proposed fault-tolerance approach is based on the concept of N-modular redundancy, Timeout mechanism and Location based dependency. N-modular redundancy means, offloading request is sent to more than one surrogate in each iteration. Surrogate will wait for the result upto the timeout period and location based dependency will help the offloadee to choose the suitable surrogate for offloading. The algorithm is executed in two steps: First, suitable Surrogates selection; Second, sending the offloading requests to the selected surrogates.

According to location based dependency our offloading technique has two variations: Location Based random Offloading- When a location is visited by the offloadee for the first time and Location Based selective Offloading- When a location is visited almost regularly.

Assumptions taken in this approach:

1. The application is need to be offloaded
2. N surrogates are available for offloading at a location
3. a person visits some places almost regularly

Terms that are used in this Fault-tolerance approach:

S_c = Surrogate count

T_i = Waiting time after sending an offloading request to surrogate(s) in each iteration

T_{max} = Maximum waiting time for offloading

(Both T_i and T_{max} are predetermined based on the nature of the offloading request)

S_{rs} = No of successful offloading responses by the surrogate

$S_{rf =}$ No of times surrogate failed

MTTF = Mean Time To Failure

S_A = Surrogate availability

(S_{rs} , $S_{rf\,,}$ MTTF, S_A can be termed as "Surrogate Parameters" or "Environmental datas")

$Req_n^{(s)}$ **=** Number of offloading requests sent to a particular surrogate

$Res_n^{(s)}$ = Number of successful offloading results from that surrogate

L_s = Link-strength(**L_s**) between the surrogate & the client

5.2.1 Location Based Random Offloading

Surrogate Selection- Before offloading surrogate selection is a very important step. We assume that the surrogates have already taken part in offloading and they have related datas accordingly. From the past offloading datas each surrogate keep track of three parameters: No of successful offloading responses (S_{rs}), No of times it failed(S_{rf}) and it's own MTTF(Mean Time To failure) value. From these parameters surrogate will calculate it's availability (S_A) = f(S_{rs}, S_{rf}, S_{MTTF}) and this availability value of each surrogate will be shown as an offloading indicator. When a location is visited for the first time, offloadee will select the surrogates based on their availability value.

Offloading: Based on the nature of the application, offloadee will fix two timeout values- Waiting time after each iteration(T_i) and Maximum waiting time($T_{max}= T_{i1}+T_{i2}+T_{i3}+...$) for offloading. Initially offloadee chooses two suitable surrogates and sends offloading request to S_c(Surrogate Count) surrogates and waits for T_i time for the result. If result does not arrive within T_i time and $T_i<T_{max}$ then offloading requests are sent to S_c+1 surrogates, but if Tmax exceeds then offloadee will start local re-execution. This implements the concept of N-modular redundancy and Timeout mechanism.

During this approach offloadee keep track of certain parameters which can be termed as "**Offloadee Parameters**" for each surrogate. Those are-

- Surrogates to which offloading requests were sent
- Link-strength(**L_s**) between the surrogate and the client
- Number of offloading requests sent to a particular surrogate($Req_n^{(s)}$)
- Number of successful offloading results from that surrogate($Res_n^{(s)}$)

From these offloadee parameters surrogate calculate the success ratio then success rate(SR) for each surrogate using the following formulas for corresponding surrogate 's' & request 'n':

$$Success\,ratio = \frac{Res_n^{(s)}}{Req_n^{(s)}} \quad (2)$$

$$Success\,rate(SR) = Success\,ratio \times 100 = \frac{Res_n^{(s)}}{Req_n^{(s)}} \times 100$$

(3)

Now, at a particular location according to this success rate, priority(P) of every surrogate will be calculated. We will take '1' as highest priority, then '2' as a priority lower than 1, so on. The surrogate having highest success rate(SR), will be assigned priority '1'. The surrogate having 2nd highest SR, priority '2' will be assigned. In this way, at a particular location all the surrogates which are available for offloading will be assigned priority according to their success rate. After calculating the success ratio, success rate and priority for all the surrogates at that location, these values are stored by the offloadee.

When a person has visited the location earlier, then offloadee will choose the surrogate using it's previously saved data of that location. So, both the historical data of the offloadee and surrogate will be utilized to select the suitable surrogate for selective offloading. This concept is shown in algorithmic form below.

Algo1:

Location Based Random Offloading(Offloadee Side)

```
RequestingOffloading($T_i$,$S_c$,$S_A$)
    1.1 Select suitable surrogates based on their availability value($S_A$).
    1.2 Send Offloading request to $S_c$(initially=2) surrogates
    1.3 Wait $T_i$ time for the result.
    1.4 If result found
        1.4.1 Save necessary information regarding offloadee parameters
    1.5 else if $T_i<T_{max}$
        1.5.1 RequestingOffloading($T_i$,$S_c$+1,$S_A$)
    1.6 else
        1.6.1 start local re-execution
```

5.2.2 Location Based Selective Offloading

It always happens that a person visits some places very frequently as well as almost every day. Suppose a person regularly goes to office, visits airport to take flights, attend meetings-conferences at particular auditoriums; Professors regularly visits colleges & so many. In all these scenarios the approach of 'selective offloading' is very effective.

Surrogate Selection: When a location is visited regularly , surrogate selection is done based on surrogate availability(S_A) and priority(P) of the surrogate. Surrogate availability(S_A) is indicated at the surrogate side and Priority of different surrogates at that location is already stored in the client side.

Offloading: In Selective offloading, initially the client will choose the surrogates having best S_A values & success ratio and send offloading request to such surrogates whose priorities are either '1' and '2'. Offloadee waits Ti time for the result. If result does not arrive within T_i time and $T_i<T_{max}$ then offloadee will increment the priority values by '1' and send requests to the surrogates having corresponding priorities. But if T_{max} exceeds then offloadee will start local re-execution. After every successful offloading offloadee will make the required changes in surrogate's priority if necessary. In this approach, probability of successful offloading is very high in each attempt, which was not there in previous approach. Here in each attempt, the client will send the offloading request to the most suitable surrogate. When a person visits a particular location regularly, this is not only a very good approach towards fault-tolerance in offloading system but also reduces the offloading execution time, thus enhancing the system performance. An algorithm of this concept is shown below:

Algo2:

<u>Location Based Selective Offloading(Offloadee side)</u>

```
RequestingOffloading($T_i$,P,$S_A$)

2.1 Select suitable surrogate based on surrogate availability value($S_A$) and priority of the surrogate(P).

2.2 Send Offloading request to surrogates having best availability and priority value.

2.3 Wait Ti time for the result

2.4 If result found

    2.4.1 update the priority values of the surrogate

2.5 else if $T_i<T_{max}$

    2.5.1 RequestingOffloading($T_i$,P+1,$S_A$)

2.6 else start local re-execution
```

5.2.3 Algorithm for Surrogate side(For both Random and Selective offloading)-

Algo3:

3.1 Periodically calculate Surrogate availability(S_A) based on surrogate parameters- S_{rs}, S_{rf}, S_{MTTF} .

3.2 Always show the availability value

3.3 Receive offloading request

3.4 If required resources are available

    3.4.1 execute the request and send back the result to the client

3.5 else offloading fails.

3.6 Record total no of offloading requests received, successful offloading results sent, no. of times it failed and MTTF value

5.3 Fault Detection and tolerance by this approach

<u>Crash Failure:</u>

Detection- If crash failure occurs then offloadee will get no result. In the above approach if the surrogate crashes then the offloadee will not be able to get the result within T_i time shown in Fig. 12.

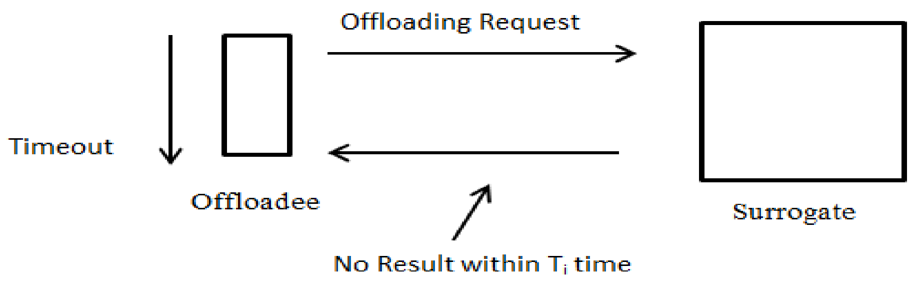


Figure 12: Occurrence of crash failure

Tolerance- If offloadee does not get the result within T_i time, then it will send the offloading request to other surrogates within the T_{max} time. It is less probable that different surrogates crash simultaneously. The probability of Offloadee getting the result from atleast one surrogate within T_{max} time is very high if redundancy concept is used. Fig.13 shows an approach to tolerate crash failure.

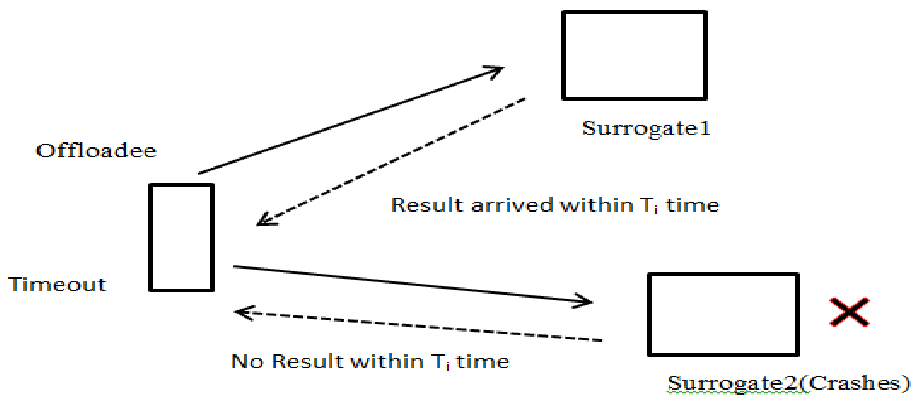


Figure 13: Tolerance of crash failure

Omission failure:

Detection- Omission failure in offloading systems, occur due to either lossy network between offloadee and the surrogate or due to surrogate unreachability. If omission failure occurs due to lossy network then offloadee may receive the result but some part of the result may be erroneous. But if the reason is surrogate unreachability then offloadee will not receive any result at all.

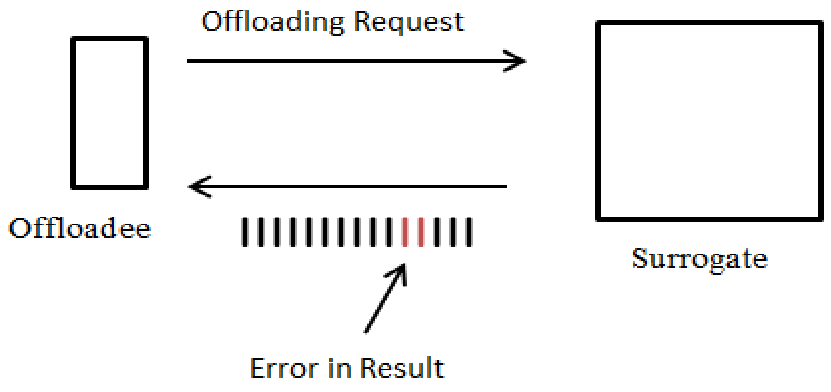


Figure 14: Omission failure detection

Tolerance- If result arrives and that is erroneous then offloadee can get the correct result from other surrogates those who have also received the offloading request according to redundancy approach. When result comes back from more than one surrogate, offloadee can also detect the error in result by majority voting. But if omission failure occurs due to surrogate unreachability, in that case failure will be handled similar to crash failure.

<u>Transient failure:</u>

Detection- Due to transient failure the system will be affected temporarily but after some time it will again start working properly. For this failure offloadee may receive an erroneous result where some of the part may be missing.

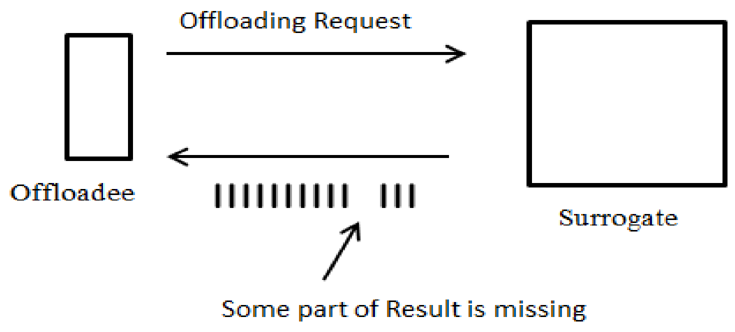


Figure 15: Transient failure detection

Tolerance- Transient failure can also be tolerated by majority voting. In this scheme, even if one surrogate crashes or gives erroneous result, offloadee will able to choose the correct result by observing the result sent by majority of surrogates.

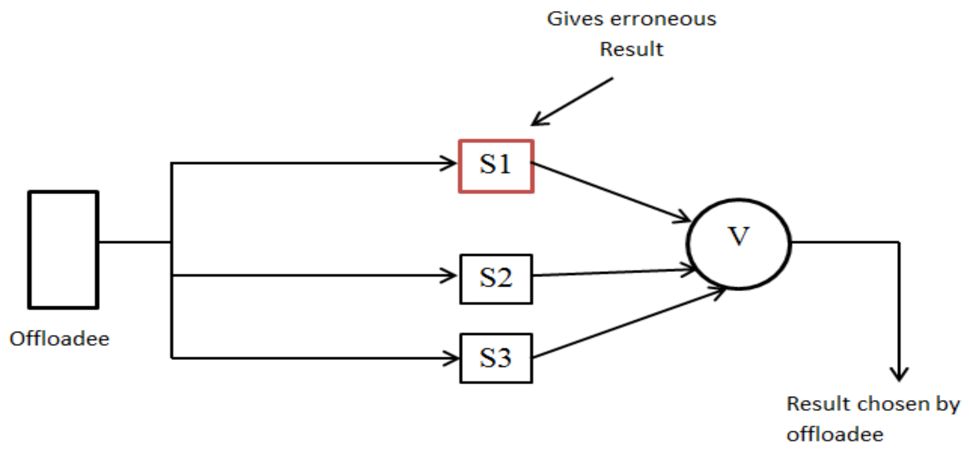


Figure 16: Tolerance of omission and transient failure using majority voting

Summary

Initially offloading system detect the faults, then fault-tolerance mechanism is employed . As this mechanism utilizes the concept of location based dependency, this helps people to choose the most suitable surrogate in a regularly visited location which reduces the occurrence of faults. Next chapter discusses the implementation details of constructing the offloading system and employing the fault-tolerance mechanism to tolerate faults.

Chapter 6: Design & Evaluation

6.1 System Setup

In this thesis, we have used a client-server based offloading model to detect different faults in offloading system and to simulate the fault-tolerant approach. Clients are typically resource-constrained devices with limited networking capability, e.g., a PDA, tablet, or cell phone. Surrogates are resource-rich devices that are willing to run programs and/or provide storage on behalf of clients. Surrogates might be a desktop PC, laptops available via the local wireless LAN(e.g., in a smart home, office, or commercial hotspot environment)or might be home PC or a commercial surrogate connected via the Internet.

For this experiment, our surrogate platform is HP Probook 4550 Series Laptops with 2.10-GHz i3 processor and 1GB of RAM. The client is a Samsung Galaxy GT-P5100 Tab running android version 4.1.2. For all experiments, mobile device and the surrogates are connected through 802.11 wireless LAN. Client programs are in android studio[41] and Surrogate programs are written in java. Experiment is executed in several locations. The link speed of the LAN connections in different locations varies between 65-75Mbps.

6.2 Design

This section comprises of two sub-sections: Designing the offloading system and then simulating the fault-tolerance approach. To design the system, first a suitable application needs to be chosen in which three options will be available for execution-

a. Local execution: The application will be executed in the mobile device only.

b. Offloading without fault-tolerance: The application will be executed in one surrogate. Here, offloadee will provide the "ip-address" and "port no" during runtime to connect with the surrogate and send the offloading request(Fig. 17).

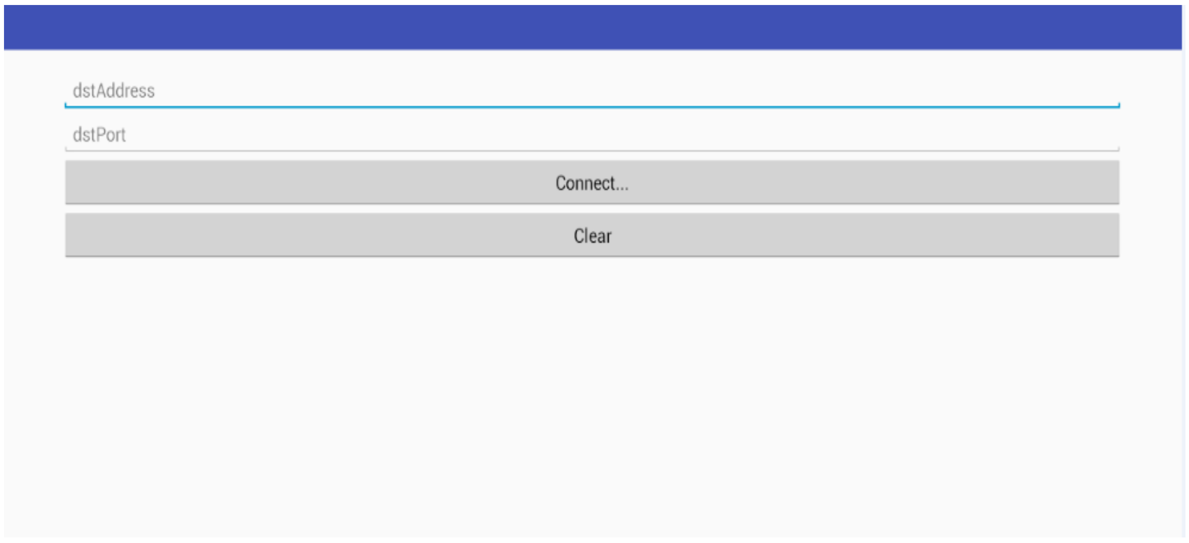
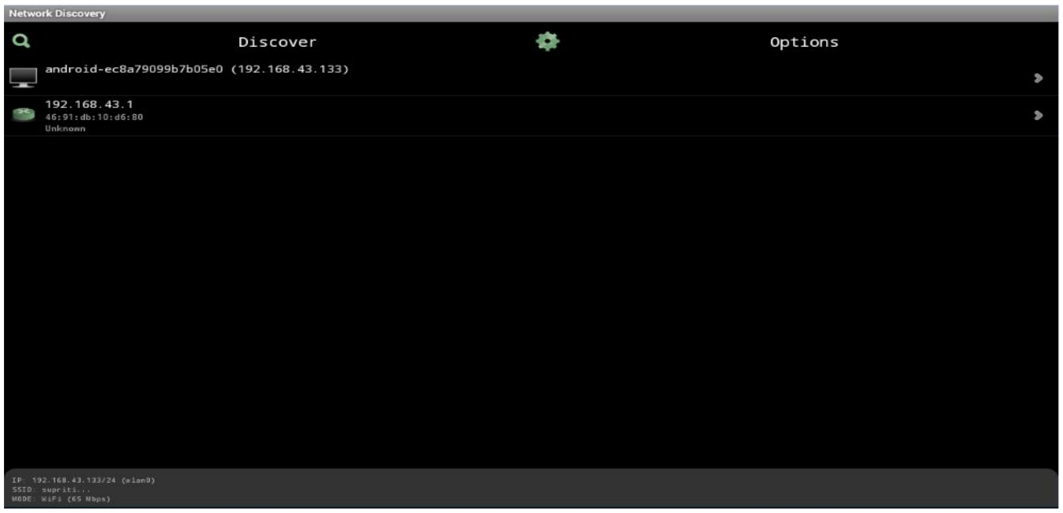


Figure 17: Connect with the surrogate

c. Offloading with fault-tolerance: The application will be executed using above fault-tolerance approach. First, “Network Discovery” will be run to find the available surrogates(Fig. 18). When the surrogates are available, offloadee will choose suitable surrogates and send them offloading requests.



```
android-ec8a79099b7b05e0 (192.168.43.133)
192.168.43.1
46:91:db:10:d6:80
Unknown
IP: 192.168.43.133/24 (wlan0)
SSID: supriti...
MODE: WiFi (65 Mbps)
```

Figure 18: Finding available surrogates

Fault-tolerance approach can be executed in three steps- First surrogate selection, then Fault-insertion and finally either location based random offloading or selective offloading.

At a location, several servers need to run the surrogate program for the offloading session. After the end of the offloading session, these surrogates will create a database to store the parameters such as- total no of offloading requests received, successful offloading results sent, MTTF value and Surrogate Availability(S_A) . Surrogate availability(S_A) will be calculated based on the first three parameters and stored in the database. When the next offloading session will start, every surrogate at that location will show their S_A value, so that offloadee can choose a suitable surrogate.

Crash failure can be injected in the surrogate by stopping it, within the offloading session. In that case, offloadee will not get the result from the surrogate within the timeout and detect the crash failure(Described in section 7.3). Omission failure can be injected by either moving the mobile device out of reach of the surrogate or disconnecting the wi-fi network. So far, in our work we have simulated crash failure and yet to observe the effect of omission and transient failure.

From random offloading execution at a location offloadee will save no. of requests sent to each surrogate($\mathbf{Req_n^{(s)}}$), No. of successful offloading result from each surrogate($\mathbf{Res_n^{(s)}}$), Link strength($\mathbf{L_s}$) and distance between the surrogate and the client. From these parameters offloadee will calculate the success ratio(SR) and priority(P) of all the surrogates at a location which will be stored in the database of offloadee. When the location is visited again, offloadee will select the surrogate with best priority(Offloadee database) and availability(Surrogate database) value.

Waiting time $T_{i,}$ T_{max} are decided according to the offloading request. The fault-tolerant offloading scenario is shown in Fig. 19.

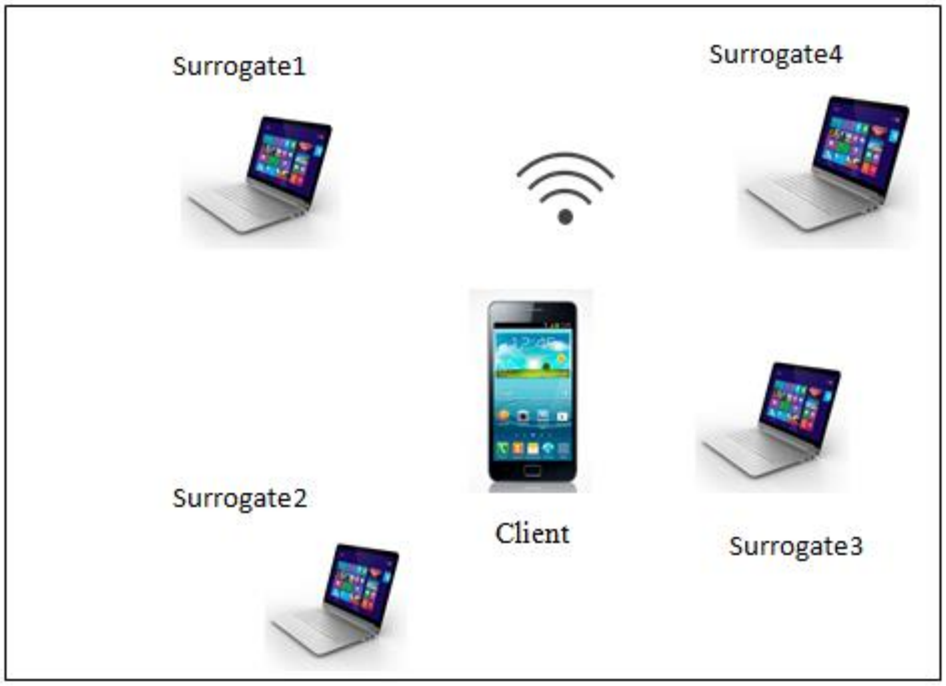


Figure 19: fault-tolerant Offloading scenario

6.3 Implementation Details

6.3.1 Π calculator

A simple Java application, Π calculator is used to evaluate the model primarily. The application is developed for calculating Π which firstly gets user's input i.e. upto which decimal places the value will be calculated and then invokes class Π for the calculation.

Some code snippets that have been used for offloading this application are mentioned below:

Android Client

1. Sender asynchronous class that sends the decimal places upto which the value of Π will be calculated

```
  private class Sender extends AsyncTask<Void, Void, Void>{
   private String message;
```

```
    @Override
    protected Void doInBackground(Void... params){
        message = textField.getText().toString();
        printwriter.write(message + "\n");
        printwriter.flush();
        return null;
     }
}
```

2. Receiver asynchronous class that receives the value of Π from the java server

```
  private class Receiver extends AsyncTask<Void, Void, Void>{
    private String message;
    @Override
      protected Void doInBackground(Void... params) {
            while (true) {
               if (bufferedReader.ready())
               message = bufferedReader.readLine();
                publishProgress(null);
             }
        }
}
```

Java Server

1. making connection between the offloadee and the surrogate:

```
ServerSocket server = new ServerSocket(4444);
 System.out.println("Server started successfully");
 Socket client = server.accept();
```

2. Receiving the decimal place upto which the value will be calculated

```
InputStreamReader j = new InputStreamReader (client.getInputStream());
 BufferedReader r = new BufferedReader(j);
PrintStream w = new PrintStream(client.getOutputStream());
String data = r.readLine();
```

```
        int digits = Integer.parseInt(data);
```

3. Calling the value of Π

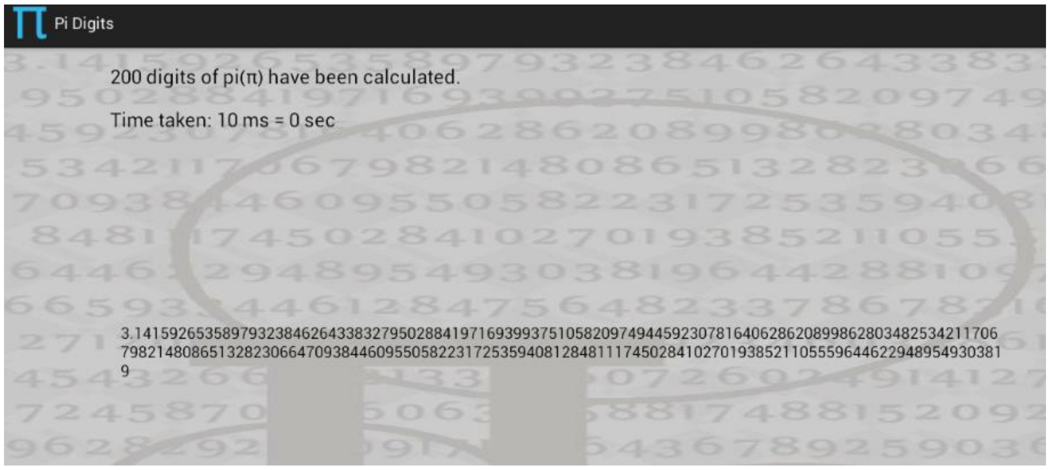
```
int max = 1000;
BigDecimal num2power6 = new BigDecimal(64);
BigDecimal sum = new BigDecimal(0);
 for(int i = 0; i < max; i++ )  {
        . . . . . . . . . . . . . . . . . . . . .
  }
    sum = sum.divide(num2power6,digits, BigDecimal.ROUND_FLOOR);
    System.out.println("value is: " +sum);
```

4. Sending the value back to the offloadee

```
BigDecimal k= sum;
System.out.println("value of picalculator upto 100 decimal places:\n\n " +sum);
w.println(k);
System.out.println("\nResult sent back to client");
```

6.3.2 Snapshots of Π calculator offloading execution

- **Execution in Offloadee only:** Calculation of Π for 200 and 1000 decimal places in the mobile device, shown in Fig.20.1 and Fig.20.2 respectively.

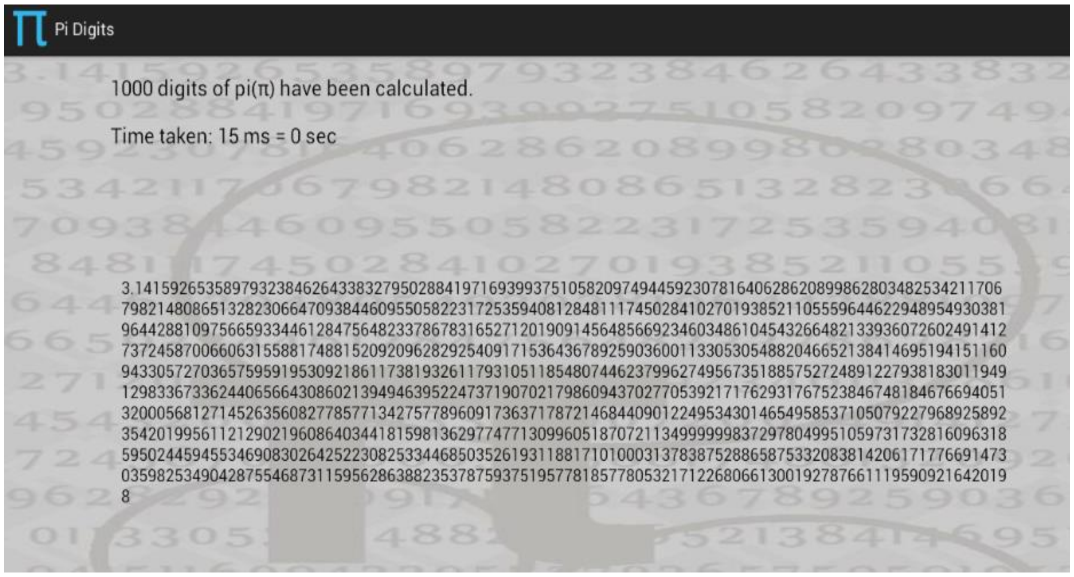


200 digits of pi(π) have been calculated.

Time taken: 10 ms = 0 sec

```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706
7982148086513282306647093844609550582231725359408128481117450284102701938521105559644622948954930381
9
```

Figure 10.1: Calculation of 200 digits of Π in the mobile device only



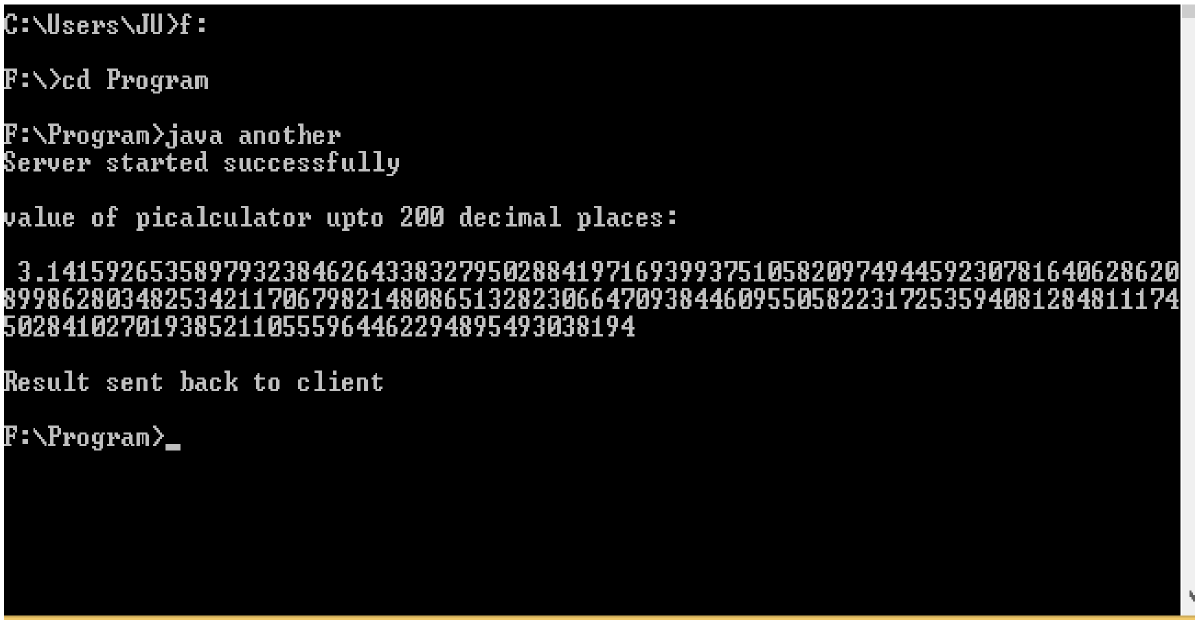
1000 digits of pi(π) have been calculated.

Time taken: 15 ms = 0 sec

```
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706
7982148086513282306647093844609550582231725359408128481117450284102701938521105559644622948954930381
9644288109756659334461284756482337867831652712019091456485669234603486104543266482133936072602491412
7372458700660631558817488152092096282925409171536436789259036001133053054882046652138414695194151160
9433057270365759591953092186117381932611793105118548074462379962749567351885752724891227938183011949
1298336733624406566430860213949463952247371907021798609437027705392171762931767523846748184676694051
3200056812714526356082778577134275778960917363717872146844090122495343014654958537105079227968925892
3542019956112129021960864034418159813629774771309960518707211349999998372978049951059731732816096318
5950244594553469083026425223082533446850352619311881710100031378387528865875332083814206171776691473
0359825349042875546873115956286388235378759375195778185778053217122680661300192787661119590921642019
8
```

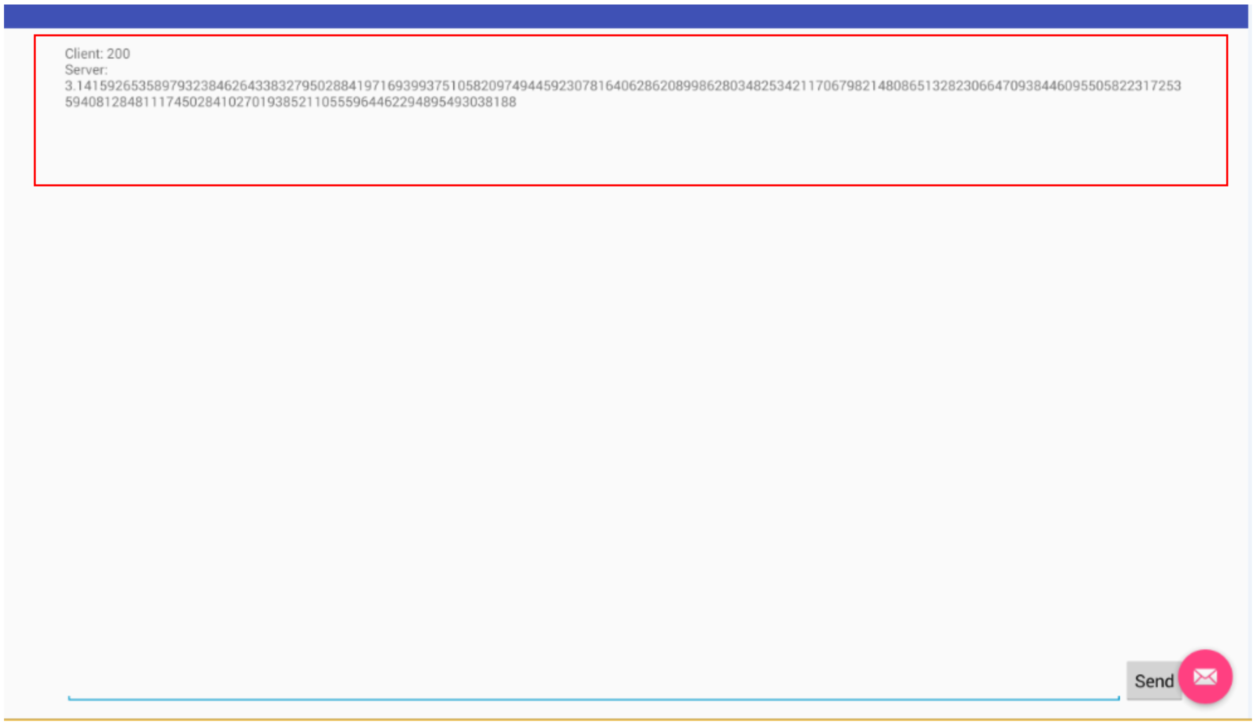
Figure 20.2: Calculation of 1000 digits of Π in the mobile device only

- **Execution in offloading:** In this case, offloadee will send the decimal place value to the surrogate, surrogate receives the value, calculate the result and send it back to offloadee. Fig. 21.1 and Fig. 21.2 shows Π calculation offloading for 200 digits and Fig. 22.1 and Fig. 22.2 shows it for 100 digits calculation.



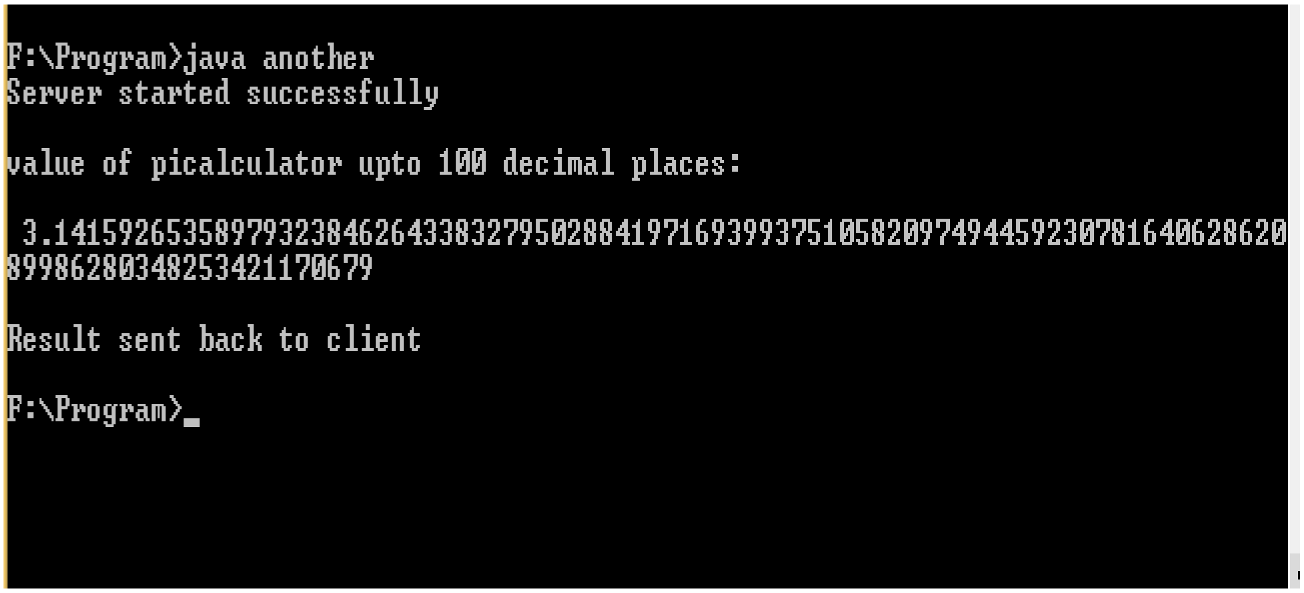
```
C:\Users\JU>f:

F:\>cd Program

F:\Program>java another
Server started successfully

value of picalculator upto 200 decimal places:

 3.14159265358979323846264338327950288419716939937510582097494459230781640628620
899862803482534211706798214808651328230664709384460955058223172535940812848111174
502841027019385211055596446229489549303819 4

Result sent back to client

F:\Program>_
```

Figure 21.1: Calculation of 200 digits of Π in the surrogate and simultaneously it sends back the result to offloadee shown in Figure 21.2



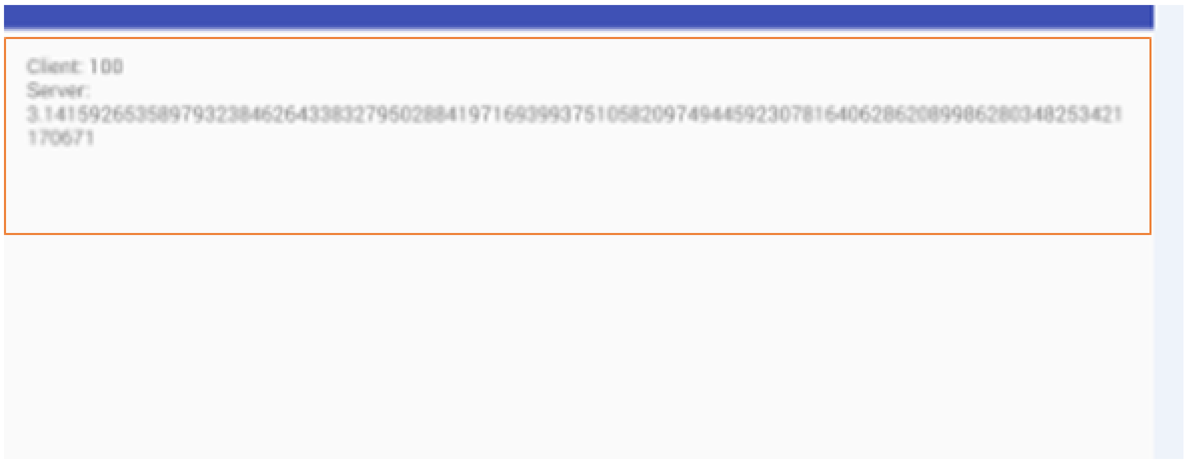
```
Client: 200
Server:
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535
94081284811174502841027019385211055596446229489549303818 8
```

Figure 21.2: Offloadee, after getting the result from surrogate(for 200 digits)



```
F:\Program>java another
Server started successfully

value of picalculator upto 100 decimal places:

 3.1415926535897932384626433832795028841971693993751058209749445923078164062862
089986280348253421170679

Result sent back to client

F:\Program>_
```

Figure 22.1: Calculation of 100 digits of Π in the surrogate which is send back to the offloadee shown in Figure 22.2



```
Client: 100
Server:
3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253421
170671
```

Figure 22.2: Offloadee, after getting the result from surrogate(for 100 digits)

6.3.3 Resource Utilization

By using an app named “System Monitor Lite[42]”, we have measured the cpu and memory consumption of the application in offloading and non-offloading method. Since, the fault-tolerance approach is not yet completed, we are unable to give it’s measurements regarding resource utilization in the following figures.

Fig.23 shows the memory usage and Fig.24 shows CPU utilization for the calculation of Π calculator, respectively. The Y-axes represent the resource usage and the X-axes represent the accuracy of Π (i.e. decimal places).

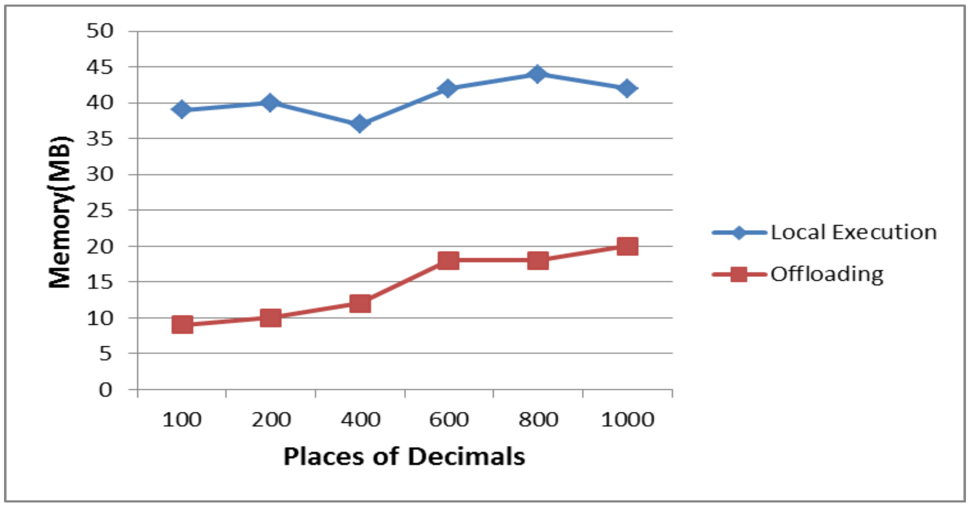


Figure 23: Memory consumption

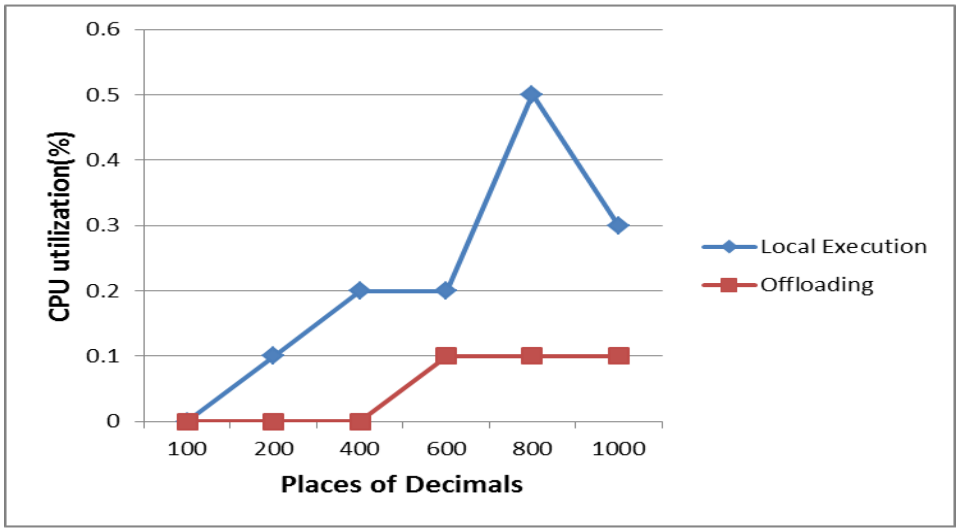


Figure 24: CPU utilization

6.3.4 Effect of Crash Failure and it's tolerance

To observe the effect of failure, faults need to be injected in the system manually; only then the performance of fault-tolerance mechanism can be evaluated properly. As discussed earlier, offloading system is susceptible of Crash, Omission, transient and security failure. Till now the effect of Crash failure has been observed in our work, effects of other failures are yet to be observed. Crash failure in offloading system generally occurs if the surrogate stops responding to the offloadee. So, this failure is injected in the system by stopping the surrogate using server. stop() command during offloading phase.

In Fig. 25, surrogate is being stopped during the calculation of 800 decimal places of Π. While offloading if crash failure occurs, system will stop working further but if fault-tolerance mechanism is implemented then failure can be tolerated after it's detection and the system will continue working. Fig. 25 shows the system performance in presence of crash failure whereas tolerance of crash failure is shown in fig. 26. For fig.26 we have assumed that crash failure is detected during the calculation of 100 decimal places and after the detection, failure is tolerated using the redundancy concept. In the following figures, X-axes represents the decimal places and y-axes represents the system performance with respect to success or failure. Success indicates '1' and failure indicates '0' performance.

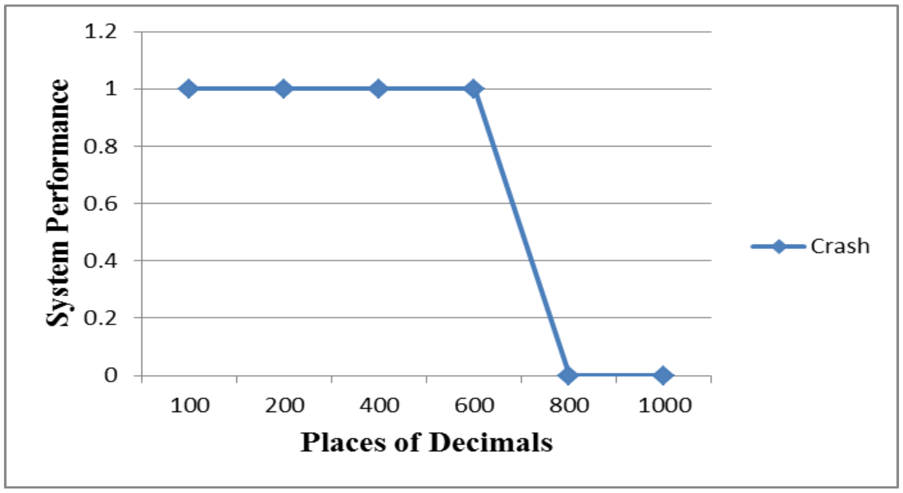


Figure 25: Occurrence of Crash Failure

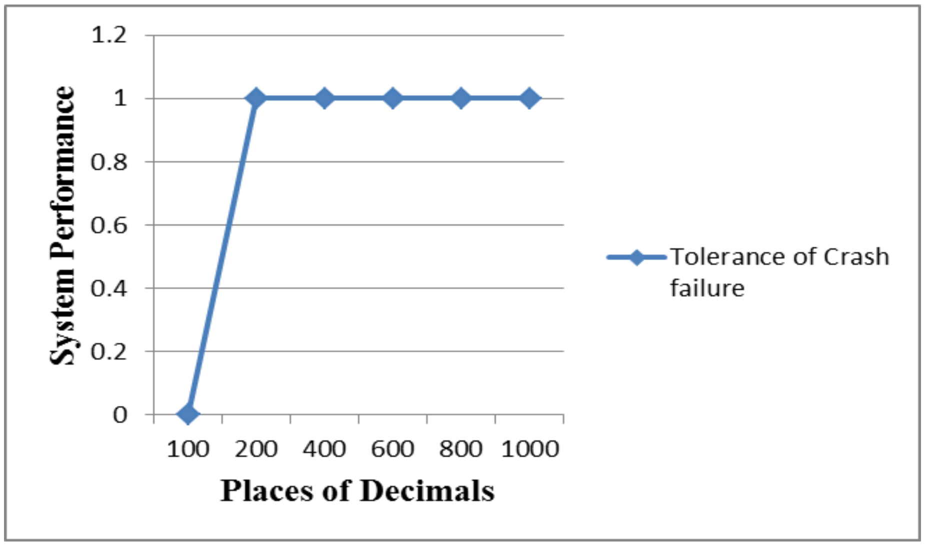


Figure 26: Tolerance of crash failure

Summary

An application of our proposed approach Π calculator is discussed in this chapter. The simulation result shows the effectiveness of offloading and crash failure tolerance capability. In the next chapter the thesis is concluded following the future scope of work.

Chapter 7: Conclusion

7.1 Summary

With the growing demand in offloading, fault detection and fault-tolerance have become an indispensable part for designing a reliable offloading system.

In this thesis, first we have detected the reasons for which offloading may fail and classified them into crash, omission, transient and security failure. After detecting the occurrence of different faults, a redundancy based fault-tolerance approach is proposed which is executed in two steps: when a person visits a location for the first time only random offloading is applicable. In that case, offloadee selects the surrogate based on surrogate availability value only and saves necessary information (e.g. available surrogates, performance of the surrogates, link strength) of that location for further processing. Now, when the person visits that same location next time his/her mobile device will utilize the previously saved informations of that location along with surrogate availability to choose the most suitable surrogate for offloading. Basically we have inserted the concept of location based dependency in offloading.

The offloading system is designed using Π calculator and evaluated in terms of resource utilization. Crash failure is simulated in the system so far to observe the performance of fault-tolerance approach.

7.2 Future Work

We plan to insert omission and transient failure in offloading system and observe the effect of fault-tolerance approach under those faults. One useful way to detect omission and transient failure is using the concept of hash code. Before sending the request to surrogate, offloadee can generate a hash code and attach it with the requesting message. When result comes back to offloadee, it will check the hash code sent during the request. If there is any change then it is quite obvious that system has faced either an omission failure or transient failure.

In this thesis we have considered the network connections, the surrogates in the offloading system to be secure and trusted. But this does not always happen in real. As described earlier, “timing attack” is one of the security issue in offloading system. So, tolerance of security failure is indispensable. In our next work, we try to incorporate mechanisms to tolerate different type of security failure in offloading system by using the concept of cryptography and trusted system.

Reference

[1] iPhone App Store. http://www.apple.com/iphone/appstore

[2] Android Market. http://www.android.com/market

[3] Raging Thunder. http://www.polarbit.com/our-games/raging-thunder-2

[4] Motorola MOTOBLUR. http://www.motorola.com/Consumers/US-EN/Consumer-Product-and-Services/MOTOBLUR/Meet-MOTOBLUR

[5] Google Maps Navigation. http://www.google.com/mobile/navigation/

[6] Health to Go. http://www.healthymagination.com/

[7] Mathew, B., Davis, A., & Fang, Z. (2003, October). A low-power accelerator for the SPHINX 3 speech recognition system. In *Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems* (pp. 210-219). ACM.

[8] Cuervo, E., Balasubramanian, A., Cho, D. K., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. (2010, June). MAUI: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services* (pp. 49-62). ACM.

[9] Kemp, R., Palmer, N., Kielmann, T., & Bal, H. (2010). Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services* (pp. 59-79). Springer Berlin Heidelberg.

[10] Goyal, S., & Carter, J. (2004, December). A lightweight secure cyber foraging infrastructure for resource-constrained devices. In *Mobile Computing Systems and Applications, 2004. WMCSA 2004. Sixth IEEE Workshop on*(pp. 186-195). IEEE.

[11] Ou, S., Yang, K., & Zhang, J. (2007). An effective offloading middleware for pervasive services on mobile devices. *Pervasive and Mobile Computing,3*(4), 362-385.

[12] Kristensen, M. D. (2010, March). Scavenger: Transparent development of efficient cyber foraging applications. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on* (pp. 217-226). IEEE.

[13] Ou, S., Yang, K., & Zhang, Q. (2006, April). An efficient runtime offloading approach for pervasive services. In *Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE* (Vol. 4, pp. 2229-2234). IEEE.

[14] Park, S., Chen, Q., Han, H., & Yeom, H. Y. (2014). Design and evaluation of mobile offloading system for web-centric devices. *Journal of Network and Computer Applications*, *40*, 105-115.

[15] Chen, G., Kang, B. T., Kandemir, M., Vijaykrishnan, N., Irwin, M. J., & Chandramouli, R. (2004). Studying energy trade offs in offloading computation/compilation in java-enabled mobile devices. *Parallel and Distributed Systems, IEEE Transactions on*, *15*(9), 795-809.

[16] Hwang, I., & Ham, J. (2014, April). Cloud offloading method for web applications. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on* (pp. 246-247). IEEE.

[17] Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., & Milojicic, D. (2003, March). Adaptive offloading inference for delivering applications in pervasive computing environments. In *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on* (pp. 107-114). IEEE.

[18] Chun, B. G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. (2011, April). Clonecloud: elastic execution between mobile device and cloud. In*Proceedings of the sixth conference on Computer systems* (pp. 301-314). ACM.

[19] Linux vserver project, available at http://www.linuxvserver.org/.

[20] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. of the 19th ACM Symposium on Operating Systems Principles*, October, 2003.

[21] Web Worker Specification, http://www.w3.org/TR/workers/

[22] Ghosh, Sukumar. *Distributed systems: an algorithmic approach*. CRC press, 2014.

[23] Wang, Q., & Wolter, K. (2014, November). Detection and Analysis of Performance Deterioration in Mobile Offloading System. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on* (pp. 420-425). IEEE.

[24] Meng, T., Wang, Q., & Wolter, K. (2015). Model-based quantitative security analysis of mobile offloading systems under timing attacks. In *Analytical and Stochastic Modelling Techniques and Applications* (pp. 143-157). Springer International Publishing.

[25] Ou, S., Wu, Y., Yang, K., & Zhou, B. (2008, May). Performance analysis of fault-tolerant offloading systems for pervasive services in mobile wireless environments. In *Communications, 2008. ICC'08. IEEE International Conference on* (pp. 1856-1860). IEEE.

[26] Avižienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, *1*(1), 11-33.

[27] Yang, K., Ou, S., & Chen, H. H. (2008). On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *Communications Magazine, IEEE*, *46*(1), 56-63.

[28] Brumley, D., & Boneh, D. (2005). Remote timing attacks are practical.*Computer Networks*, *48*(5), 701-716.

[29] Huang, Y., Kintala, C., Kolettis, N., & Fulton, N. D. (1995, June). Software rejuvenation: Analysis, module and applications. In *Fault-Tolerant Computing, 1995. FTCS-25. Digest of Papers., Twenty-Fifth International Symposium on* (pp. 381-390). IEEE.

[30] Cotroneo, D., Natella, R., Pietrantuono, R., & Russo, S. (2011, November). Software aging and rejuvenation: Where we are and where we are going. In*Software Aging and Rejuvenation (WoSAR), 2011 IEEE Third International Workshop on* (pp. 1-6). IEEE.

[31] Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M., & Chen, X. (2012). Comet: Code offload by migrating execution transparently. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)* (pp. 93-106).

[32] Van Moorsel, A., & Wolter, K. (2006). Analysis of restart mechanisms in software systems. *Software Engineering, IEEE Transactions on*, *32*(8), 547-558.

[33] Sheahan, R., Lipsky, L., Fiorini, P. M., & Asmussen, S. (2006). On the completion time distribution for tasks that must restart from the beginning if a failure occurs. *ACM SIGMETRICS Performance Evaluation Review*, *34*(3), 24-26.

[34] Asmussen, S., Fiorini, P., Lipsky, L., Rolski, T., & Sheahan, R. (2008). Asymptotic behavior of total times for jobs that must start over if a failure occurs. *Mathematics of Operations Research*, *33*(4), 932-944.

[35] Liu, J., & Lu, Y. H. (2010, October). Energy savings in privacy-preserving computation offloading with protection by homomorphic encryption. In*Proceedings of the 2010 international conference on Power aware computing and systems, HotPower* (Vol. 10, pp. 1-7).

[36] Liu, J., Kumar, K., & Lu, Y. H. (2010, August). Tradeoff between energy savings and privacy protection in computation offloading. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design* (pp. 213-218). ACM.

[37] Ou, S., Yang, K., & Hu, L. (2007, November). Cross: a combined routing and surrogate selection algorithm for pervasive service offloading in mobile ad hoc environments. In *Global Telecommunications Conference, 2007. GLOBECOM'07. IEEE* (pp. 720-725). IEEE.(20)

[38] Clausen, Thomas, and Philippe Jacquet. *Optimized link state routing protocol (OLSR)*. No. RFC 3626. 2003.

[39] Johnson, David B. "The dynamic source routing protocol for mobile ad hoc networks." *draft-ietf-manet-dsr-09. txt* (2003).

[40] Rebeiro, C., Mukhopadhyay, D., & Bhattacharya, S. (2015). An introduction to timing attacks. In *Timing Channels in Cryptography* (pp. 1-11). Springer International Publishing.

[41] Android Studio. http://developer.android.com/sdk/index.html

[42] System Monitor Lite.
https://play.google.com/store/apps/details?id=com.cgollner.systemmonitor.lite&hl=en/