Dissertation

zur Erlangung des akademischen Grades
des Doktors der Naturwissenschaften

(Dr. rer. nat)

Restart in Mobile Offloading

eingereicht
am Institut für Informatik
des Fachbereichs Mathematik und Informatik
der Freien Universität Berlin
von

Qiushi Wang

Berlin, 2015

Gutachter:

Prof. Dr. Katinka Wolter
Department of Computer Science
Freie Universität Berlin

Prof. Dr. Aad van Moorsel
School of Computing Science
University of Newcastle upon Tyne

Tag der Disputation: 20. November 2015

Eidesstattliche Erklärung

Ich versichere, dass ich die Doktorarbeit selbständig verfasst, und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit hat keiner anderen Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass bei Verwendung von Inhalten aus dem Internet ich diese zu kennzeichnen und einen Ausdruck mit Angabe des Datums sowie der Internet-Adresse als Anhang der Doktorarbeit anzugeben habe.

I hereby declare to have written this thesis on my own. I have used no other literature and resources than the ones referenced. All text passages that are literal or logical copies from other publications have been marked accordingly. All figures and pictures have been created by me or their sources are referenced accordingly. This thesis has not been submitted in the same or a similar version to any other examination board.

Berlin, den September 12, 2015
Qiushi Wang

Abstract

Offloading is an advanced technique to improve the performance of mobile devices. In a mobile offloading system, richer functional applications are developed by migrating heavy computation from resource constrained thin clients like mobile devices to powerful cloud servers. Although today heavy computation can also be completed by most clients, it consumes much time and energy. Therefore, offloading heavy computation can sometimes effectively accelerate the processing speed and prolong battery lifetime for mobile devices.

To guarantee the completion of such offloading tasks, strong connectivity between client and the server is essential. Generally, wireless networks are utilized to support the connectivity. However, a wireless network sometimes is not sufficiently qualified to provide a reliable communication between clients and servers. Task completion can be delayed by congestion or packet loss in the network, and execution continuity is always interrupted by network failures. In brief, the intermittent connectivity in wireless network constrains the advantage of offloading applications. To deal with this problem, restart is an efficient method that can reduce the task completion when the network quality is bad. In mobile offloading systems, besides retrying the same offloading task, jobs can be locally restarted and completed in the client device itself. Adaptively selecting the right option and automatically restarting at the appropriate moment can balance out undesired effects. While the mechanism of restart itself is very simple, deciding when to apply it is not easy at all.

In this thesis, the optimal moment to launch restart is identified according to different metrics. For reaching a balance between energy consumption and throughput, the best restart time should be able to acquire a trade-off between the two aspects. Under unstable network qualities, the optimal moment is derived to minimize the expected task completion time. In addition, in order to reducing the burden of server, multiple clients should not launch restart at the same time.

Zusammenfassung

Offloading ist eine fortschrittliche Technik um die Performance mobiler Endgeräte zu verbessern. In einem mobile offloading System werden funktional umfangreiche Applikationen durch Migration rechenintensiver Prozesse von ressourcentechnisch eingeschränkten Klienten, wie Mobilgeräten, auf leistungsstarke Server in der Cloud entwickelt. Auch wenn heute auch aufwendige Berechnungen auf Mobilgerten durchgeführt werden können, kostet dies viel Zeit und Energie. Deshalb kann das Auslagern dieser Berechnungen manchmal effektiv die Ausführung auf Mobilgeräten beschleunigen und die Batterielebensdauer der Mobilgeräte verlängern.

Um das Beenden von offloading nutzenden Prozessen zu garantieren, ist eine sehr gute Verbindung zwischen Klient und Server erforderlich. Üblicherweise werden kabellose Netzwerke zur Bereitstellung der Verbindung genutzt. Ein kabelloses Netzwerk kann jedoch unter Umständen nicht dafür geeignet sein, eine zuverlässige Kommunikation zwischen Klient und Server zu bieten. Die Vollendung der Berechnungen kann durch Überlastung oder Verlust von Paketen im Netzwerk verzögert werden und die Ausführung wird ständig durch Netzwerkfehler unterbrochen. Kurz, die instabile Verbindung in kabellosen Netzwerken schränkt die Vorteile von Applikationen, die mobile offloading nutzen, ein. Das Neustarten der Berechnungen ist eine effektive Methode um dieses Problem anzugehen und kann die Ausführungszeit bei schlechter Verbindungsqualität verringern. In mobile offloading Systemen kann neben dem Versuch die selbe Berechnung erneut auszulagern auch die Berechnung auf dem Mobilgerät selbst neu gestartet und durchgefhrt werden. Adaptiv die passende Option auszuwählen und das automatische Neustarten im richtigen Moment kann unerwünschte Effekte ausbalancieren. Während der Mechanismus des Neustartens an sich sehr einfach ist, ist die Entscheidung wann er angewandt werden soll ganz und gar nicht einfach.

In dieser Arbeit wird der optimale Zeitpunkt zum Neustarten bezüglich unterschiedlicher Metriken identifiziert. Um ein Gleichgewicht zwischen Energieverbrauch und Durchsatz zu erreichen, sollte die beste Zeit zum Neustart einen Kompromiss zwischen diesen beiden Aspekten bieten. Unter stabiler Netzwerkqualität wird der optimale Zeitpunkt durch die Minimierung der erwarteten Ausführungszeit abgeleitet. Um die Belastung des Servers zu reduzieren, sollten zusätzlich nicht mehrere Klienten gleichzeitig ihre Berechnungen neu starten.

Acknowledgement

When I nearly completed this thesis, I just found how precious time is. Even before I really feel it, four years have already run away. At that moment, I was so nervous that I did not know what I have done in this period. Then I make myself calm down and carefully recollect my beautiful memories in these years.

Before I came to Berlin, I had never lived in a foreign country for such a long time. So on the flight from Beijing, I was always thinking how it could be. I expected to learn, to experience and also to contribute something here. Today, when I recall my life in Berlin, I am very grateful that all my expectations have been achieved. And suddenly I find that so much I love this city, this University and my lab. Although I have to leave, the memorable days I spent here will stay in my heart forever. Sincerely appreciation would be given to my families, my colleagues and my friends.

Many many thanks to my supervisor Prof. Dr. Katinka Wolter. In my four years research, she gave me quite a lot of help to improve my capability and expand my horizontal. I still clearly remember how I was amazed and surprised when I saw her detailed comments on my first paper sentence by sentence. In my heart, I deeply admire her professional research skill and positive work attitude. Without her direction, I cannot imagine to reach today's success. Thanks a lot for providing me a chance to study here and leading me on the research road.

I also appreciate Prof. Dr. -Ing. Jochen Schiller to accept me in his research team and give me a nice office for my research. His warm heart and sense of humour always give me a deep impression. Talking with him makes me feel relaxed and comfortable. He is a successful and respectable leader in out team. A lot of thanks will be given to him for supporting me every time when I need it.

I have to say that I am really a lucky man, because I met my wife before I started this long unknown research life. She is my strongest supporter whenever I encounter difficulties or suffer failures. In the four years, my wife sacrificed a lot for me. She spent much time to relieve and inspire me when I experienced upset. My wife is also another type of guide in my life. Her patience and generosity help me to grow as a man with more responsibility and confidence. Every morning when I wake up and receive her greeting, I know that I am not fighting alone. We are a team. With her accompany, I never feel lonely in a foreign country. With her encouragement, I always feel full of energy to complete my work. With her trust, I believe that we will embrace a happy life in the future! I also feel very lucky to own some tiny pretty partners like Baoli and Fatboy. They increase much fun in my simple life.

Thanks to my parents! They have spent almost everything they can provide to bring me up. I really do not know how to pay them back. I feel very regret to leave home for such a long time. Although I cannot take care of them when they need me, they always forgive me. They often tell me my happiness is their happiness. I hope my success today can bring them honour and pride.

Thanks to Mart Griera Jorba and Joan Martnez Ripoll. They helped me build the framework of the mobile offloading system. Thanks to Fabian Reimeier, he helped me translate the abstract into German. Thanks to Bodo Riediger-Klaus, Ren Skillen and other staff in the IT support team of our department. They always responded me quickly once my desktop or the server had problems. And I would thank to all my colleagues. In my four years research, they give me many support and helpful suggestions. Thanks to Dr. Philipp Reinecke and Dipl.-Inform. Matthias Wählisch for their selfless support during the research and discussions. I also would like to thank Prof. Dr. Marcel Kyas, Stephanie Bahe, Dr. Emmanuel Baccelli, Dipl.-Inform. Norman Dziengel, Dipl.-Inform. Oliver Hahm, Dipl.-Inform. Alexandra Danilkina, Dipl.-Ing. Stephan Adler and all the members of CST for their kindly daily help, which makes me quickly get involved into the research life in FU-berlin. Thanks to my Chinese colleagues. I earned a solid friendship. I will give my thanks to Yubing Zhao, Yuan Yang, Huaming Wu, Yi Sun, and Tianhui Meng for their help and memorable time.

My study is under the financial support by China Scholarship Council (CSC) and carried on at the institute of computer science of Freie Universität Berlin. And thanks to the members of ”China Scholarship Council Stipendiaten Vereinigung”. We shared our happiness and never feel lonely. I am very grateful to the Chinese Embassy in Germany, particularly to the staffs in the Education Section. I enjoy the wonderful life in Germany with their good care and support.

In the end, I wish all my families, my friends and me a happy life! I enjoy the four years time in Berlin!

Contents

Part I

Introduction

Chapter 1

Basic Concepts and Problems

1.1 Mobile Offloading

In recent years, a large number of applications have been developed for mobile devices. Obviously, many of these colourful applications have added convenience to our lives. For example, tourists will never worry about getting lost in an unfamiliar city, various navigation applications can provide the precise route information about any destination a tourist may want to visit. All powerful functions provided by these intelligent mobile applications originate from a significant improvement of mobile devices. Both the hardware processing rate and the flexibility of operating systems are able to undertake some heavier computation which could previously only be run on desktops or servers.

However, although the invention of more advanced mobile devices has improved their speed, they are still unable to compete with their desktop and server siblings. The constraints are obvious. The implementation of compute intensive applications is still limited by the constraint of the mobile device hardware, for example the long time operation of microchips cannot be sustained by low capacity batteries. The limited battery capacity prevents a long run time of some computate intensive applications (like image processing or gaming). At present, reducing the energy consumption of mobile applications remains a tough challenge. The low capacity battery is also the bottleneck of developing more attractive applications for mobile devices[108].

Moreover, even though smart mobile devices have undergone a fast development, the constraint of limited battery capacity is not merely a temporary technological deficiency but is intrinsic to mobility [122]. The trend in development of mobile device architectures and batteries shows the difficulty to overcome this constraint in the near future. To deal with this problem, Offloading is one of the popular techniques. It migrates heavy computation to remote servers through a wireless network.

In recent years, cloud computing has seen a significant development. And it is still one of the most important research directions of future computer technologies. At the same time, with the invention of more advanced wireless networks such as WiFi, 3G and LTE, mobile devices have become the most widespread terminals to access cloud services. Users can easily reach an abundance of resources: files, music and videos in the cloud instead of being restricted by the size of device storage. Therefore, the concept of offloading to the Cloud is employed to handle performance problems [49].

By migrating heavy computation to resourceful cloud servers, mobile devices can overcome the limitation of deficient resources. Offloading can not only save energy but also accelerate the execution and thus provides a better user-experience. Another benefit of connecting mobile devices with the cloud is providing better reliability. Maintaining the back-up files in cloud storage avoids data loss due to system collapse or any other disaster. In addition, as the mobile device is quite portable, ubiquitous computation services can be obtained by using the powerful computation capability of remote servers through mobile devices.

Ideally, offloading does not require a mass data transmission between mobile clients and remote servers. It only needs an overhead of several Megabits to migrate the executing thread [34] or the application state [39] from mobile terminals to remote servers and then get back the results. This delay will not impair the application performance.

1.2 Challenges in Task Completion

By migrating heavy computation to powerful cloud servers, mobile offloading systems can circumvent constraints of the client hardware, e.g. high energy consumption microchips and low capacity batteries. But the smooth offloading of computation from mobile devices to cloud servers depends on a fast and stable network connection, which guarantees seamless communication. The success of offloading computation relies on this stable wireless network to provide safe and reliable connection support. The network state plays an important role in the offloading system. To some degree, the performance of offloading computations is directly affected by the connection quality.

With advanced wireless networks such as WiFi, 3G or LTE, setting up a reliable connection between mobile terminals and remote servers seems possible in principle. Unfortunately, the quality of a network is not constant across space and time. Even though wireless networks have undergone a striking development, the transmission quality still varies. It is impossible to persistently provide a reliable connection, which guarantees the success of offloading. Consequently, the execution of an offloading task may suffer from long delays or even failures in the network. In [150] the impact of unreliable network connections on mobile offloading has been experimentally confirmed. In addition, using wireless connectivity demands a lot of energy [28, 31]. The limited battery capacity often cannot support the mobile device to wait an unpredictable time for the network to recover, which may take very long.

As the remote cloud servers of large enterprises are supported by better back-up and protection schemes, comparing with the frequently fluctuated wireless network condition, they have comparably high-availiability. In that way, the wireless network becomes the major cause of bringing unstability to offloading systems. However, even though the remote cloud servers of large enterprises are mostly maintained by better back-up and protection schemes, sometimes they still experience long downtime [29, 46]. Therefore, these cloud servers cannot be seen as perfectly reliable systems. This can be another cause of the connection failure.

The above situations indicate that connection failures cannot be avoided completely during the execution of offloading, which means failure handling schemes are indispensable. Some efficient schemes are introduced in the next section.

1.3 Performance Improvement with Restart

In computer and network systems, when a task is subject to failures or unpredictable delays, aborting the previous try and launching a new process can speed up the task completion. This mechanism is generally called restart and widely used in preventive maintenance [142], software rejuvenation and fault tolerance [47]. Examples of such tasks include randomised search algorithms, distributed data queries and data transmission through unreliable network connections. These tasks are also widely spread in Cloud computing and Cyber-physical systems. As introduced in [98], restart allows to express an efficient tradeoff between the average and the variance in the time a task will take.

Usually, there are two major restart schemes to handle the failure problem in the mobile offloading system. The first one is halting in 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 wrapped in failure, a long time is wasted for waiting and accompanied with large energy consumption. As currently the function of mobile devices has been improved remarkably, executing the offloading tasks locally in the mobile device is the other method to handle failures. Although it costs more time and battery energy, the continuity of application execution is maintained. To avoid ambiguity, in this thesis, restart and re-execution are deemed as the same meaning.

The question arises when to launch the local re-execution. Immediately re-executing the predetermined offloading task locally as soon as the wireless connection breaks only adapts to the previous extreme scenario of a very long repair time because the local re-execution is not cost-effective. Comparing with running in remote servers, more time and energy has to be provided by the mobile device to complete the same intensive computation.

In case 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. In order to provide a balanced performance in both the scenarios, combining the two schemes with a timeout scheme is a reasonable method.

In addition, jointly using offloading and local restart also constitutes another kind of hybrid restart scheme. This scheme works as launching multiple restart at intervals until the offloading task is completed. First the job attempts to restart with offloading. Then, if the number of restarts exceeds the threshold, the job is completed locally. The main problem that must be solved is the optimal limiting threshold on the number of allowed restart tries.

Generally, restart is a simple yet efficient solution, which can be applied if the probability distribution of a task completion time exhibits high variance. If we use the expected task completion time as metric to evaluate the performance under different thresholds, utilizing some stochastic analysis models or heuristic iterative algorithms we can theoretically identify the optimal threshold.

1.4 Contributions

The main contributions of this thesis help to address failure handling requirements in the mobile offloading systems. The first part of this thesis provides the background and related work about the performance improvement of the mobile offloading system. The second part experimentally confirms the impact of network quality on the running of mobile offloading system. The third part proposes some restart-based schemes to deal with the problem of unstable network quality. The efficiency of these schemes is confirmed by either experiments or simulations.

In the following we give a short summary of these contributions. The organisation of this thesis is introduced chapter-by-chapter in section 1.5, below.

Part I: Introduction

As a novel innovation, mobile offloading still has no structure standard or an explicit definition. It covers a wide range of concept referring to Cloud Computing, Mobile Communication and Distributed Computation. In order to generalize some common properties of the system structure, we make a survey about existing mobile offloading platforms. Although several different techniques have been utilized to build the mobile offloading platform, for instances virtual machine [124], program partitioning [104], code offload [39] and so on, their fundamental work flows are identical.

A model-based analyse method can provide valuable insights into the system performance. There is no doubt that a lot of researchers have contributed models in this area. We draw some representatives to illustrate the inspiration of the model proposed in this thesis. The merits and drawbacks of those representatives are also compared.

Part II: Unstable Network Quality

In order to demonstrate the impact of unstable network quality on the mobile offloading system, we have to solve several problems: First, the quality of the network must be assessed, second, the variation of the network quality must be monitored based on which then an estimate of the optimal restart timeout is computed. We assume that the system performance is positively correlated with the network quality, and use the task completion time as a metric to evaluate system performance. Although the energy consumption is also very important to evaluate performance, as introduced in [28, 39, 99], we are not able to easily determine energy usage and fall back to task completion time. We state that for our purposes there is a sufficiently strong correlation between energy consumption and task run time. To monitor the variation in network quality we dynamically build a histogram of the task completion time which provides a good and timely estimate of its distribution. We propose a method to periodically update the histogram.

Part III: Failure Handling with Restart

If we only consider local execution after a restart. The key problem behind restart is when to launch it. There clearly exists a tradeoff between the cost of local or remote retry and waiting for the offloading to succeed. We mathematically derive conditions for applying local restart and the optimal timeout based on a greedy method and we propose a dynamic online scheme to determine whether and when to launch a local restart. Then, we adapt the optimal restart time at run time in order to account for the variation of the network quality. An automated adaptive restart scheme by jointly using offloading and local restart is proposed.

In addition, a static method is also proposed to find the optimal timeout when to restart locally by analysing the system performance using stochastic models. We used SAN(Stochastic Activity Network)[121] models to simulate the execution of offloading systems and computed three metrics: Instability, Energy Consumption and Throughput to evaluate the performance of the offloading system. It has been shown through simulation that the optimal timeout changes when the quality of the network deteriorates. We confirm those previous simulation results by analysing experimental data and use an integrated method by combining the experimental test-bed and a simulation model to find out the optimal waiting time for launching the local re-execution.

1.5 Organisation of the Thesis

In the first part of the thesis we generally introduce the background of this thesis and some related work. Part I has the following structure:

In **Chapter 1**, I describe some elements of our story: where it happens, who is involved and how it develops. Mobile offloading system is the research filed of this thesis. It provides the stage for the two main characters: network quality and restart schemes. The story originates from that the unreliable network brings a risk to the smooth execution of offloading task. The process of this story mainly focuses on how to adapt the restart scheme to the mobile offloading system and reduce the risk by optimization.

In **Chapter 2**, a survey on some typical structures of mobile offloading systems is presented. Then, the related stochastic models of performance analysis for the computer system and network are reviewed. After that, several popular methods used in automatic adaptive control are briefly introduced.

The second part of the thesis concerns the impact of varying network quality on the mobile offloading system and is structured as follows:

In **Chapter 3**, an experiment is introduced to study the impact of packet loss and delay on the task completion time. The distribution of task completion time under an unreliable network exhibits a heavy-tailed profile. The experimental results confirm the need for local restart.

The third part of the thesis illustrates the implementation of the restart scheme and the evaluation of its efficiency. Part III is structured as follows:

In **Chapter 4**, we describe the mathematical derivation of a condition which is used to determine whether and when to launch a restart. The condition is further extended to adapt for the mobile offloading system. The criteria of whether and when to launch a local restart is derived.

In **Chapter 5**, we introduce the structure of our program engine implemented with some offloading applications. Optical Character Recognition (OCR) is exploited as the main sample application in the experiments to verify the unreliable network and evaluate the restart performance.

In **Chapter 6**, a description of the proposed SAN models is presented. We use the models to simulate the execution of the offloading system. Three metrics, instability, energy consumption and throughput are introduced. They are defined to evaluate the performance of various local re-execution start moments. After normalization, the three metrics are synthesized to compare the overall performance.

In **Chapter 7**, a modular based system model is proposed to analyse the impact of multiple users on the cloud server in the mobile offloading system. The congestion occurs in the server side when the task arrival rate in the client side increases. To deal with this problem, we proposed a congestion avoidance scheme by using local execution to complete parts of the offloading tasks.

In **Chapter 8**, the dynamic restart scheme is introduced. A dynamic histogram is designed and exploited to estimate the theoretic criteria of launching local restart. The efficiency of single local restart and optimal adaptive restart is illustrated using experiments.

In **Chapter 9**, the main part of the thesis is concluded with a summary. The contribution of restart to reduce the impact of unreliable network in mobile offloading system is emphasized. In the end, we provide the outline of our future research directions.

Chapter 2

Related Work

Although, a lot of surveys about mobile offloading and cloud computing exist such as, [83, 52, 30, 44, 10, 78, 63, 109, 5], they are either too general as including a wide range of related works, or too specific as forwarding to a direction which is not concerned in this thesis. Therefore, in this chapter we introduce some representative work on mobile offloading system. We provide a view of the cloud computing from the perspective of mobile offloading. After that, we refer to some popular models which are used to analyse the performance of mobile offloading system and cloud computing. Through these models, readers can easily understand the inspiration and origination of the research approach utilized in this thesis. Then, we summarise the features of data transmission in the public Internet. Since the communication between clients and servers is the key element which decides the success of completing the offloading tasks, understanding the features of data transmission can give an impression of the importance of applying methods to deal with the unstable network quality.

2.1 Typical Mobile Offloading System

2.1.1 Historic Background

Powerful distributed systems as in Cloud computing aim at turning computing as utility into reality [35]. Recently, as mobile devices have become the most popular clients to access Cloud services, the concept of pervasive computation services was proposed to integrate mobile devices with Cloud Computing. As soon as the concept of cloud computing was proposed, it has attracted attentions to integrate mobile devices within the cloud.

Thin clients using a remote infrastructure for compute-intensive tasks have already been seen as a method for addressing the challenges of distribution and mobility as in pervasive computing [123]. To reach this destination, mobile offloading has been developed as to merge Cloud computing and mobile computing. Offloading the computation from smart phones to remote resourceful cloud servers has also been rediscovered as a technique to enhance the performance of mobile applications, while reducing the energy consumption [88, 152, 113, 111].

In general, it makes sense to not always offload, instead, it should be performed optionally considering some conditions. For example, if there is no internet connection, it should be possible to execute the whole application locally in the mobile device. Based on some parameters, such as the access time to the server, the internet bandwidth, the amount of data to transmit, the remaining battery lifetime or the estimated execution cost of a potentially offloadable computation, the offloading component has to decide whether to migrate the movable part of code. Offloading has to decide, based on different parameters (e.g. wireless network quality, device compute capability and remaining battery capacity), which parts of the movable components should be migrated to execute in remote servers.

In order to identify the movable components in mobile applications, Partitioning is one of the important and fundamental step to implement mobile offloading. Partitioning tries to reasonably separate the mobile application into several components and classify them into two categories: movable and local. Some of them can be executed in remote servers are marked as movable, and the others can only be run locally in mobile devices are marked as local. A large number of contributions exist in this area[87, 61, 104, 89, 147, 55, 54, 60, 112].

2.1.2 Existing Offloading Systems

Under the strong motivation of making mobile devices fast and energy-efficient, mobile offloading as a concept has been around for more than a decade. Several methods (CloneCloud, MAUI, Cloudlets, among others) have been proposed to support the seamless use of augmented computation to a mobile device. Most of them are architectures that want to provide a comfortable development environment for the programmers, who want to implement their mobile applications with offloading.

Research in offloading methods can be divided into three main directions [49]: client-server communication, virtual machine migration and mobile agents.

We will now discuss related work in all three areas.

1. **Client-server communication:**
   Communication can be supported by pre-installation of the application in both the mobile client and the server. In this case one can benefit from existing stable protocols for process communication between mobile and surrogate devices. This is the basis for the systems in [50, 13, 94, 67, 43, 77].
2. **Virtual machine migration:**
   Offloading can be implemented as the migration of the complete virtual maching executing the application. The most fascinating property of this method is that no code is changed for offloading of a program. The memory image of a mobile client is copied and

transferred to the destination server without interrupting or stopping any execution on the client. Although this method has clear advantages as it avoids having two versions of a program, it requires a high volume of transferred data [34, 39, 66, 124, 110].

3. **Mobile agents:**
   Scavenger [84] introduced a framework that partitions and distributes heavy jobs to mobile surrogates in the vicinity rather than to cloud servers. Offloading to more than one surrogate is the merit of this framework. In [153], the authors proposed a seamless offloading service for the client-surrogate structure. The effectiveness and efficiency of the novel service are validated by both the experiment in a real-life mobile Internet application scenario and the simulation in large-scale and more dynamic mobile network environments.

Few of the above approaches tackle the problem of when to offload and which communication partner to choose. In [8] the authors designed a Markov decision process to find the optimal aging control policies, which decide when to connect to the server and which network link to use.

2.1.3 Prototypes

Among the three categories, the virtual machine migration is the most popular one and represents the future development trend. The great advantage of virtual machine is the high consistency of the two version applications in the mobile side and the server side. Since nearly no modification is required to move the application from the mobile device to the server, the burden on the programmer is reduced. Although the client-server communication supports a more robust link between both sides, the application has to be pre-installed. The main drawback of mobile agents is the expensive cost of agent management and application synchronization. For understanding the detailed properties of each category, we make a deep exploration into the specific structures. As the virtual machine has attracted most attention, we first list four sample structures belonging to this category below:

CloneCloud boosts unmodified mobile applications by offloading the right portion of their execution onto device clones operating in a computational cloud [34]. It proposes a very interesting idea: to continuously have a synchronized copy of the mobile device contents in the cloud in order to offload operations like file searches or virus scans.

MAUI achieves the two benefits of maximizing the potential for energy savings through fine-grained code offload, while minimizing the changes required to applications [39]. It applies the managed code environments to dynamically partition programs according to the methods profiling and serialization of an application.

VM-Based Cloudlets exploit virtual machine (VM) technology to rapidly instantiate customized service software on a nearby cloudlet and use that service over a wireless LAN [124]. Due to the physical proximity between cloudlets and mobile clients, the one-hop network latency brings the advantage of transient connections at the cost of a large-scale implementation with high price.

Mobicloud treats mobile devices as service nodes and organises them in an ad-hoc network. Each node is mirrored in the cloud and used as a virtual component to provide computation service [66]. In addition to providing traditional computation services, Mobicloud enhances communication by addressing trust management, secure routing, and risk management issues in the network.

Although the client-server communication is less elastic than the virtual machine, the well supported Application Programing Interfaces (APIs) reduce the complexity of system implementation. Some representative system structures based on the mode of client-server communication are listed here.

Cuckoo is a complete framework for computation offloading for Android, including a runtime system, a resource manager application for smart phone users and a programming model for developers [77]. The Ibis High Performance Programming System [143] is used as the basis for Cuckoo's communication component.

Chroma is a tactics-based remote execution system, which makes perfect partitioning decisions at runtime [13]. Tactics is a compact declarative form which captures the knowledge about an application relevant to remote execution. Chroma improves application performance by automatically utilizing extra resources in an over-provisioned environment.

Hyrax allows client applications to conveniently utilize data and execute computing jobs on networks of smart phones and heterogeneous networks of phones and servers by porting Hadoop [129] and MapReduce [42] to run on Android smart phones [94]. Hyrax explores the possibility of using a cluster of mobile phones as resource providers and shows the feasibility of such a mobile cloud.

These platforms can be considered as the prototypes of a general and standard mobile offloading structure. We believe that the mobile offloading system is still in early stage. With the invention of more intelligent mobile devices, the mobile offloading structure will naturally adapt itself to the development. There is no doubt, more novel structures will also be proposed. In fact, if some uniform standards can be formulated to define either the offloading work flow or the basic elements of the system structure, the development of mobile offloading will be facilitated.

2.1.4 Model-based Analysis Techniques

All offloading systems mentioned so far may suffer from poor network conditions and the application of well-designed fault-tolerance methods is necessary. Restart is suitable since it is a simple and popular recovery scheme to mitigate network failures. Markov chain models and Laplace transforms have been developed to analyse the performance of restart for improving the expected task completion time [11, 85, 126, 86, 17]. The above analytical work strongly supports the efficiency of restart if the best restart timeout is known. A fast method based on iteration theory to identify the optimal restart time has been presented in [98]. The algorithm has been improved in [141, 140, 142]. It has been tailored for Internet applications in [117]. Based on the assumption that successive runs are independent, [91] has described a provably optimal policy to minimise the expected completion time. In [76] prior results on fixed restart policies have been extended to more efficient dynamic restarts by using predictive models to provide solvers with a real-time ability to update beliefs about run time.

In addition, for improving the performance of a mobile offloading system, the impact of network deterioration can also be mitigated by either optimizing the network configuration or developing new management schemes for the client. For instance, Bulut and Szymanski introduced an efficient algorithm to increase the offloading ratio based on the density of user data request frequency [27]. They measure how much offloading can be achieved with different number of Wifi access points (APs), and formulate the optimal APs deployment as an Integer Linear Programming problem. The simulation results indicate that their greedy heuristic based algorithms can closely approach the optimal solution. On the server side, the authors of [81] proposed new design techniques for a storage system which can minimize the effect of run-time server failures on the availability of intermediate data. The authors define intermediated data as the enormous amount of distributed data generated by parallel dataflow programs. Although these data are short-lived, they are critical for completion of the job and for performance improvement of the run-time server.

Generally, mobile offloading systems rely on an always-on connectivity to provide a scalable and high quality mobile access. Intelligent Radio Network Access (IRNA) provides another solution for this critical challenge. In [80], Klein, etc. proposed a heterogeneous access management framework by exploiting the IRNA-based contest information of mobile clients.

In this section, we have provided a introduction of the development of the mobile offloading system. First, the origination of the offloading concept is reviewed. Second, some representative system architectures are depicted and compared. Last, we present advanced techniques utilized to handle failures caused by unstable network connections between clients and servers.

2.2 Models for Performance Analysis

For analysing the performance of communication or computation systems, one of the popular ways is to use stochastic models. These models have also been proved as an efficient method to emulate the system operation and estimate the system performance for the mobile offloading system. In this section, we first review some previous works which are related to the model-based performance analysis in mobile offloading system. Then, we introduce a specific model named Stochastic Activity Network(SAN) and its representative applications. For its flexibility and powerful functionality, SAN is used in this thesis. At last, we review some representative models which are popular in the performance analysis of Cloud Computing.

2.2.1 Models for Offloading

In previous research, models from the Markov family have been widely used. Gabner, etc. introduced an analytical Markov model to investigate component migration performance [55]. The proposed model can be used to determine the optimal policy for component configuration and to quantify the gain that is obtained via reconfiguration. In[105, 103], a semi-Markov chain model was used to express the performance of a fault-tolerant offloading system. The factor of time is analysed in these models, and it considers only waiting for resuming the offloading task until network recovery. A monitoring technique based on a Markov chain model was proposed in [106] to predict resource states. In general, the resource states at different moments are considered independent and identically distributed. The next state of a given resource is hardly related to previous states. So to some degree, the accuracy of monitoring based prediction is difficult to be verified.

However, these models can only analyze the execution time, and merely consider the influence of wireless network factors like Mean Time to Failure(MTTF). For making an intelligent offloading decision under different runtime environments, the authors of [104] utilised a dynamic multi-cost graph to model the costs of an application in terms of its component classes, including CPU cost, memory cost and communication cost. The program is adaptively partitioned based on a Heavy-Edge and Light-Vertex Matching (HELVM) algorithm, which is used to coarsen the multi-cost graph. In [61], the Fuzzy Control model was employed for making offloading decisions. The Fuzzy Control model includes a generic fuzzy inference engine based on fuzzy logic theory, and decision-making rule specifications provided by system or application developers. When the current system and network conditions match any specified rule, an offloading action is triggered.

Partitioning the application reasonably into two parts, local and offload, is another important factor to make the offloading decision. The authors of [147] showed how to use parametric program analysis to deal with the program partitioning problem of computation offloading. The optimal partitioning problem is modeled as a min-cut network flow problem with run-time parameters. The cost analysis obtains program computation workload and communication cost expressed as functions of run-time parameters, and the parametric partitioning algorithm finds the optimal program partitioning corresponding to different ranges of run-time parameters. In [59], Giurgiu pursued a flexible architectural to model the impact of the application distribution scheme, the workload size and intensity, and the resource variations of the mobile-cloud configuration on the interaction response time of the application. The correlation between these factors are also analysed.

2.2.2 Stochastic Activity Network

Markov chains are the foundation of various evolved models which have been widely used in the performance analysis [82]. Nearly no corner of the research area where time variability is concerned has not been covered by Markov chain models. Petri-net [127, 93] is a popular evolution of Markov chain models. In order to represent the timeliness and parallelism of the system in a stochastic setting, several extensions have been proposed like GSPNs (General Stochastic Petri Nets) and SRNs[102, 139]. SAN was a further extension to Petri-nets which have been extensively used for performance modeling to analyze computer and communication systems [121].

In order to obtain realistic composite performance and availability measures, Stochastic Reward Nets (SRN), as well as continuous time Markov chains were used in [92] to construct models for the performance evaluation. The authors analyse the performance changes that are associated with failure recovery behaviour. As the modelling techniques in [92] yield accurate results of prediction, SRN has been demonstrated as a powerful modelling tool for performance, availability and reliability analysis. SRN has also been widely used in computer and network systems, for example wireless communication system [137, 138], clustered system [148] and system rejuvenation policies [120].

SAN was defined with the purpose of facilitating unified performance or dependability evaluation [121]. The essential elements of SANs are timed and instantaneous activities. The execution process of the modelled system can be easily represented by them. A gate in a SAN model, supports a more flexible definition of the enabling and completion rules. It is conveniently used to represent the selection among various operation options. In brief, SAN is a general modelling technique that can be easily used to analyse the performance of computer systems [41]. Many researchers have provided contributions about how to use SRN in different parts of the computer system, such as mobile software system [22] and system security evaluation [51].

2.2.3 Specific Models for Cloud Computing

Although handling diverse client demand and managing unexpected failures without degrading performance are two key advantages of cloud service, the expanded scale and complexity of a cloud system increases the difficulty to evaluate a cloud service quality. If many configurations, workload scenarios, and management methods are included into analysis, the measurement-based evaluation of cloud service quality is expensive. In [57, 56], a general model-based approach was proposed to analyse an end-to-end performance of a cloud service. This approach divides the overall model into sub-models and obtains the overall solution by iteration over individual sub-model solutions. Assembling these sub-models yields a high fidelity model which is tractable and scalable.

Horvitz, etc. introduced a Bayesian based model to predict the run time of problem solvers [65]. From the perspective of theoretic analysis, the authors concluded when the expected time remaining of the current instance is greater than the expected time of the next instance, as defined by the background marginal model, it is better to cease activity and perform a restart. Chickering, etc. investigated a Bayesian network based approach to represent the conditional probability distributions with a decision-graph, which can provide a better solution for a greedy search on the distribution [33].

In cloud computing networks, edges and nodes have various capacities due to failure, partial failure or maintenance. In [90], an algorithm was introduced to estimate the performance of a cloud computing network under maintenance budget with nodes failure. The authors constructed a network model to describe the flows and capacities in terms of minimal paths. The quantity of data sent from the cloud to the client under the maintenance budget and time constraints is used as a metric to evaluate the maintenance reliability.

In this section, we enumerated some representative stochastic models used for analysing the mobile offloading system and the cloud computing system. These Markov based models have been confirmed as an efficient method to study the impact of the changing runtime environment on the system operation. Inspired by the previous modelling work, we design our own stochastic models for analysing the mobile offloading system under the unstable network.

2.3 Features of Date Transmission

In the last decade, there is no doubt that the Internet was dominated by the TCP/IP protocol. As an important protocol suite, it provides a reliable data transfer service to support colourful Internet applications. Although the TCP/IP protocol guarantees the successful of data communication between end hosts, the frequently changing throughput of packets transmitted brings a big challenge to network design and management. Sometimes, bursts of traffic lead to severe congestions in the network nodes. In this case, the user experience is badly impaired by the long response delay. Therefore, closely monitoring the network state at run time is significant for the network management. Measuring the round trip time (RTT) of packet transmission and comprehensively analysing the RTT statistics are important to identify some unexpected state changes in the network.

2.3.1 Round Trip Time

The round trip time of a TCP packet is defined as the elapsed time between the instant a packet is released by the source to the instant the corresponding ACK packet is received by the source. This interval includes propagation, queuing, processing and other delays at routers and end hosts. It is well known that the performance of a TCP flow is affected by its RTT. As described in [68], a TCP flow's throughput is inversely proportional to its RTT. The understanding of the phenomenon which causes the delay in RTT is essential for balancing the traffic in the network. Both the data rates realized by individual flows sharing a link and the utilization of Internet links are dramatically affected by the distribution of RTTs. Thus, the RTT distribution plays an important role in buffer provisioning, configuration of active queue management and detection of congestion unresponsive traffic.

Previous studies have shown that the RTT distribution is very environment dependent [26, 114, 24, 100]. Routing changes or queueing delay variations in some nodes on the path are the two main factors which vary the RTT that a connection experiences. As the data backward path may differ from the forward path, the traffic in only one direction (either source-to-destination or destination-to-source) should be looked at to infer the RTT distribution. In addition, the RTT distribution at a link depends on the geographical location of each connection end-points. Therefore, it is expected that different links can have significantly different RTT distributions. By analysing the packet delay, [9, 19] have found high variability of the RTT distribution with strong short and long term correlations [135]. Another interesting property of RTT is the Self-similarity, which was observed in [20].

2.3.2 Variability of RTT Distribution

A large number of researches have contributed to profiling the characteristic of RTT and to emulate its distribution [101, 95, 21, 75, 25, 155]. We reviewed some representatives of the work here to help the reader understand the importance of RTT and how to estimate its properties.

A comprehensive analysis method was proposed in [48] to identify the changes of network states. The authors carry out measurements of TCP traffic round-trip delay in the Ericsson Corporate Network. They find the round-trip delay can be well approximated by a truncated normal distribution. Between these network state changes, the period has been confirmed to be stationary with the same round-trip delay distribution. In [7], the authors studied the degree of variability in TCP RTT by passively analysing traces of over 1 million TCP connections between sources at a large campus and more than 250,000 remote destinations. Their results indicate that variability of RTT values within a single TCP connection is much wider than reported in previous studies [53, 71, 75, 95, 156]. Their observations also exhibit that the range of RTTs experienced by TCP segments is extremely large. In addition, the authors conclude that connections with smaller min RTTs see a greater variability in RTTs.

Biaz and Vaidya confirmed the coefficients of correlation between the variations of RTT and the variations of congestion window size is often weak [15]. Because RTT measured by TCP is imprecise and bears a high random component independent of the actions (increasing or decreasing the load) of the sender. When a TCP sender increases its load, the RTT may either increase or also decrease due to several factors. The most important factor is that RTT observed by a given TCP connection is dependent on other traffic carried by the network, not just on the actions of this TCP connection, especially when this TCP connection consumes a small fraction of the available bandwidth. The results in [15] argued that the Congestion Avoidance Techniques, proposed in [23, 151, 69], cannot be very useful to draw conclusions about the cause of packet loss.

2.3.3 Measurement Methodology

A large number of measurement methodologies have been proposed to estimate the distribution of RTT, we only introduce some of them which have been widely cited. In [75], a passive method was proposed. Verification experiment shows that about 90% of the passive measurements are within 10% or 5ms, whichever is larger, of the RTT that PING would measure. The authors also investigate the RTT variations in the granularity of hourly timescale. By analysing a large number of TCP trace data in the USA and Europe, they conclude that the RTT distribution at the day time is larger than in the night. The authors of [125] investigated the performance of TCP as a delayed feedback system. The authors use three methods (Syn based, Flight and Rate Change) to estimate RTT and showed that all three methods provide a consistent description of the RTT distribution. The validity of using fluid models in TCP analysis is confirmed.

In [18], the authors proposed a diffusion model based on Markov processes to reproduce the distribution of RTT. The transition probabilities and the stationary distributions of the model parameters are approximated with a mixture of Laplace distributions and Normal distributions respectively. As the RTT data fabricated by the diffusion model has a high degree of similarity compared with the data collected through practical experiments, [18] contradicted some previous opinions that the Internet is unable to be modelled. Another similar methodology was introduced in [6], a Discrete Approximation model is proposed for RTT (DA-RTT) emulation. Using the measurement data from a large university campus as input, a synthetic TCP traffic is generated from the model. By performing experiments on real test-bed with the synthetic traffic, the experiment results demonstrate that the simple DA-RTT model can closely represent the per-connection RTTs in the original traffic.

A distributed, adaptive and light-weight algorithm, Vivaldi, was introduced in [40] to predict the communication latency between the Internet hosts. Vivaldi assigns synthetic coordinates to hosts and measures the distance between the hosts. The authors propose a model, height vectors, to represent the Internet in simple geometric spaces. From their experiment results, they conclude that the cause of the access link latency may attribute to three aspects: 1) queuing delay, as in the case of an overload cable line, 2) low bandwidth, as in the case of Digital Subscriber Line (DSL), cable modems, or telephone modems, 3) the sheer length of the link, as in the case of long-distance fiber-optic cables. In [62], a new method, King, was introduced to estimate the latency between arbitrary end hosts by using recursive Domain Name System (DNS). Compared to previous approaches, three advantages of King are: 1) no additional infrastructure deployment is required, 2) end hosts do not have to agree upon a set of reference points, 3) the estimates are based on direct online measurements rather than offline extrapolation. In the experiments, the authors find that even small errors in estimates translate to large relative errors. One possible source of error is application-level latency introduced by the name servers themselves, while processing the queries of clients.

In this section, we reviewed work related to the analysis of data transmission in the Internet. The definition of RTT and its usage are introduced. We also refer to some efficient methods used for measuring the data of RTT. From the large number of measurement data, the changing network state over time is confirmed. The unstable network quality is reflected by the variability of RTT distribution.

2.4 Summary

In this chapter, we provide the background introduction for this thesis. First, we reviewed mobile offloading systems. By tracing back the origination of the offloading concept, readers can understand the intention of utilizing mobile offloading. Some popular existing offloading platforms and techniques are also listed, and their advantage and drawbacks are compared. Second, we introduced some related work on stochastic models. The applications of these models in analysing the mobile offloading system and Cloud computing are also presented. The reason of using Markov based models in this thesis is explained by showing the merits and efficiency of these models. Last, we presented the variable delay in data transmission. Many researchers have proved that an unstable network state leads to an unpredictable change of the RTT distribution, and the quality of end-to-end communication is badly impaired. To deal with this problem, some efficient methods are required. As shown in the next chapter, restart is one of them.

Part II

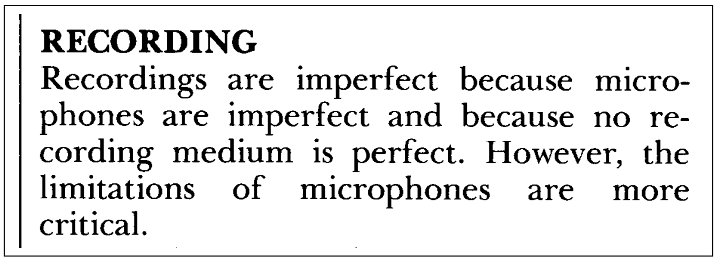
Unstable Network Quality

Chapter 3

Experimental Analysis of Response Time

In order to observe and analyse the impact of unstable network on the mobile offloading system, we design an experiment. Using the experiment we show the variability of the system performance in the presence of an unstable network. Given that the task completion time consists of the remote execution time and the data transmission time, generally, the remote execution time is assumed constant for a given task and device and delays are added by data transfer. In particular, we assume that the task completion time on the mobile device and on the cloud server can be different, but both will be more or less constant for identical tasks at different times. The offloading completion time varies greatly because data transmission times are not the same. The impact of heavy load on the server is not considered here.

In the remainder of this chapter, we first introduce the sample application, Optical Character Recognition(OCR) which is implemented in our mobile offloading engine. The details of the mobile offloading engine is introduced in Chapter 5. We use OCR here for demonstration purposes. Then we experimentally demonstrate how system performance varies over the day due to changing load in our wireless network. The task completion time is described by fitting a distribution to selected subsets of the data. This shows that the variance in the task completion time distribution increases significantly for certain subsets of the data. At last, in order to explore the impact of unreliable network on the public Internet, we conduct another experiment to demonstrate the varying delay of data transmission between a normal Linux desktop to the servers of some IT magnates.



RECORDING
Recordings are imperfect because microphones are imperfect and because no recording medium is perfect. However, the limitations of microphones are more critical.

Figure 3.1: The image to be recognised

3.1 Experiment Configuration

Offloading can be beneficial if two conditions hold. First, the task must consist of heavy computation requirement and, second, a small amount of data must be transmitted between the mobile device and the server. An application which meets both requirements is Optical Character Recognition (OCR), but there are many more. OCR is a method to recognise the characters on a binary image with optional polygonal text regions. Generally, the recognition algorithm consists of three steps: 1) The layout of the image is analysed to find some baselines of the text region. 2) The text region is chopped into components based on the gaps in the baselines. 3) Each component is recognised as several characters by comparing its shape with a trained database. For details of OCR the interested reader is referred to [130].

All three steps of OCR require heavy computation. A series of complicated edits to the image like rotation, segmentation and comparison has to be done. Performing those tasks on the mobile device consumes a lot of energy. For the powerful remote server, energy-usage is not a critical metric. In addition, most text images can be stored in small files of at most a few kilobytes. So the amount of data to transmit from the mobile device to the remote server is small. But still the time needed for the transmission depends on the quality of the network connection.

For the experiments a mobile phone (Samsung GT-S7568, Android 4.0) and a server (4 cores: Intel Xeon CPU E5649 2.53GHz) have been used. The mobile phone is placed in a dormitory room and connects to the Internet through Wifi (54Mbps provided by a local Telecom operator). The server is in the lab of the university campus and connects to the Internet through a LAN port of 100Mps. We have used the Linux command ”traceroute” to track the route from the mobile phone to the server. Normally, the route passes 12 hops to reach the destination, and the total round-trip time is around 82ms. The offloading engine as introduced in Section 5.2 includes an Android Application (App) for the mobile client and a website project for the server. In our experiment, the Tesseract OCR Engine [3] has been implemented in both parts of the offloading engine. An image (1160×391px, 8.1 KiB) with a rectangle text region, as shown in Fig. 3.1, is used for image recognition. Only 100 Bytes are used to represent the decyphered words.

Completion of an offloaded OCR task can be divided into three phases: 1) the Android ap-

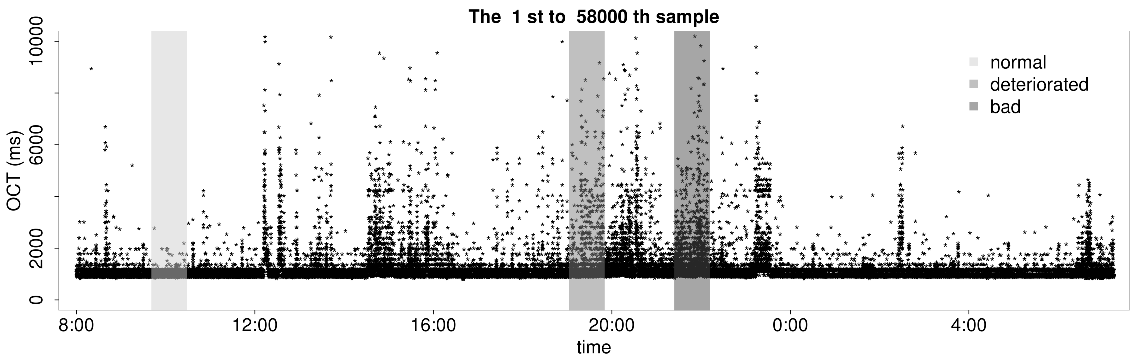


Figure 3.2: Scatter plot of all OCT samples

plication transmits the image from the mobile device to the server, 2) the words on the image are recognised using the OCR engine in the server, and 3) the mobile device receives and displays the result from the server. The Offloading Completion Time (OCT) is the time needed to complete the three steps. The same offloading task has been repeated more than 58 000 times in approximately 24 hours in order to observe OCT under the different network conditions. The results are stored in a text file in the mobile device. The memory of the mobile phone used for caching is cleared after the task completion and reused again in the next new task.

In addition, we conducted a different experiment where the image recognition is performed in the mobile device. We call it local execution, as all the processing steps (e.g. analysis, chopping and recognition) are completed by the mobile device itself. The completion time is called Local Completion Time (LCT). The same image Fig. 3.1 is repeatedly recognised 8 400 times by the local execution. In the next subsection, we will show that although local execution is slower than offloading, it is more stable than the latter.

3.2 Experimental Results

Fig. 3.2 shows a scatter plot of all data of the entire 58 000 samples over a 24-hour period starting at 8am on 14th January 2014. Under the assumption of a constant processing time, a large total completion time can be attributed to a long transmission time, i.e. poor network performance. The majority of the samples fall into the range between 980ms and 1380ms, corresponding to the 0.05 and 0.75 quantile of all the samples. Obviously the distribution of the sample values is not identical at different times. While we do not know the reason for systematic changes in network transmission times, there are clearly several types of typical behaviour that should be distinguished.

We have selected three subsets of our observations as indicated by the shaded areas in Fig. 3.2,

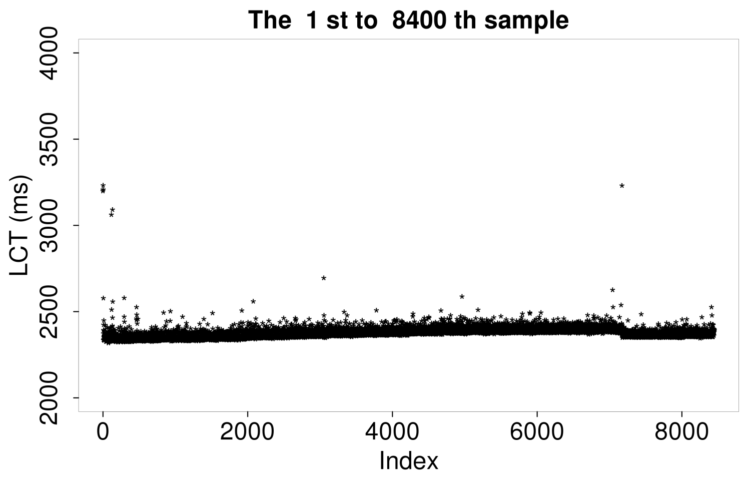


Figure 3.3: Scatter plot of all LCT samples

each containing 2000 samples, which corresponds to a time window of 40 minutes each. The number of samples is enough to decently fit a distribution and capture one type of network behaviour, the normal, the deteriorated and the bad state.

Table 3.1 shows the mean, the quantiles and the variance of the three subsets. In the *normal* subset the mean completion time has a low variability, as the 0.9 quantile is only 15% higher than the mean. It is also worth mentioning, that for the given application and setup offloading normally takes in total only half as long as local computation, because remote servers are much faster than mobile devices.

Table 3.1: Statistics of completion times (msec)

	Normal	Deteriorated	Bad	LCT
mean	1191	1618	2183	2377
0.6-quantile	1171	1466	2075	2382
0.9-quantile	1358	2595	3027	2411
0.99-quantile	1575	5495	7514	2480
variance	14496	80 5861	1680265	1249

The completion time measurement of the local computation (LCT) are shown in Fig. 3.3. Local computation is usually stable, with very few outliers. Most samples fall into a narrow range between 2338ms and 2411ms, corresponding to the 0.05 and 0.9 quantile.

In summary, in the best case offloading can provide a solution in approximately half the time needed for local processing. On the other hand, local execution times are very stable, albeit longer than processing using offloading, which suffers from high variability and, hence,

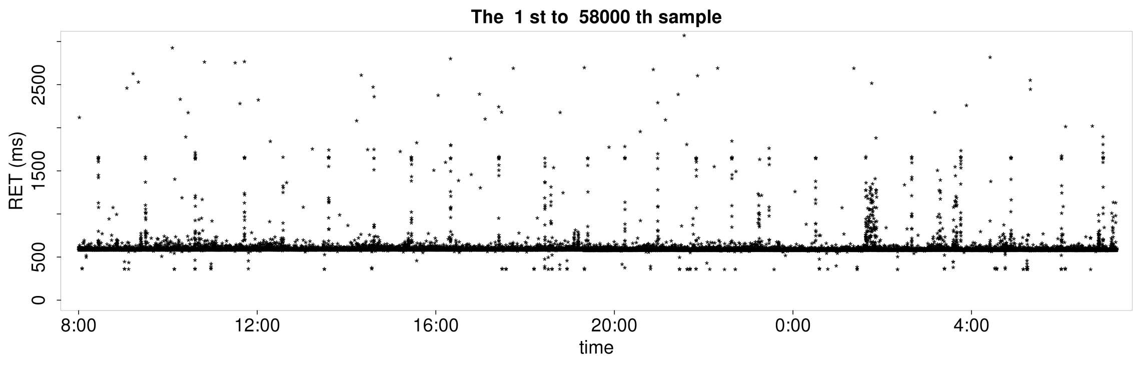


Figure 3.4: Scatter plot of all RET samples

sometimes takes very long.

The task completion time consists of two parts: data transmission time and remote execution time (RET). Before analysing the impact of network quality, we first observe the stability of the remote server. Fig. 3.4 shows a scatter plot of remote execution time T_{RET} of the same 58000 samples.

It seems that the outliers in Fig. 3.4 appear periodically. However, directly calculating the autocorrelation function of the T_{RET} samples cannot find the period, because the samples are not a time series. Since the task completion is not only controlled by the server but also affected by the network, the server throughput is reflected by T_{OCT} not T_{RET}. In order to evaluate the service rate of the remote server, we use the mean remote execution time of all tasks completed in one minute, $\overline{T}_{\text{RET}}$, as metric. Actually, $\overline{T}_{\text{RET}}$ is a time series which reflects the variation of server performance with time.

$$\overline{T}_{\text{RET}} = \frac{\sum_{i=1}^{N} T_{\text{RET}}^{i}}{N} \tag{3.1}$$

N is the number of tasks completed in one minute, and T_{RET}^{i} is the remote execution time of each task.

Fig. 3.5 shows the autocorrelation function plotted against lag (in minutes) for $\overline{T}_{\text{RET}}$. Obviously, the plot shows significant autocorrelation at the lag which corresponds to an hour (lag 60 = 60 minutes). As the autocorrelation persists over all lags, it is a clear indication that $\overline{T}_{\text{RET}}$ has a periodicity of one hour. The period of performance degradation rightly corresponds to that of the server regular rejuvenation. Although the rejuvenation process impairs the system performance temporarily, the long-haul stability of the server is guaranteed. In addition, comparing the value of outliers in Fig. 3.2 and Fig. 3.4, the long remote execution time has a limited impact on the offloading task completion, because the 0.99 quantile of $\overline{T}_{\text{RET}}$ is 727ms, which is much smaller than that of T_{OCT}, i.e. 4264ms.

Fig. 3.6 shows the box-plot of data transmission time T_{Tran} in each hour. Very roughly speaking, it seems like the network degrades most in the early afternoon and in the evening. We do not try to explain this, as finding the cause for network delays is not the scope of this thesis. Rather we argue that offloading, as well as a local restart make sense for certain network conditions and since we rightfully assume that network conditions change over time a sliding window (three shaded areas in Fig. 3.2) estimate is needed and appropriate.

It should be noted that on the average, even in poor network condition the offloaded task completes faster than the one that is computed locally. However, for the bad network period,

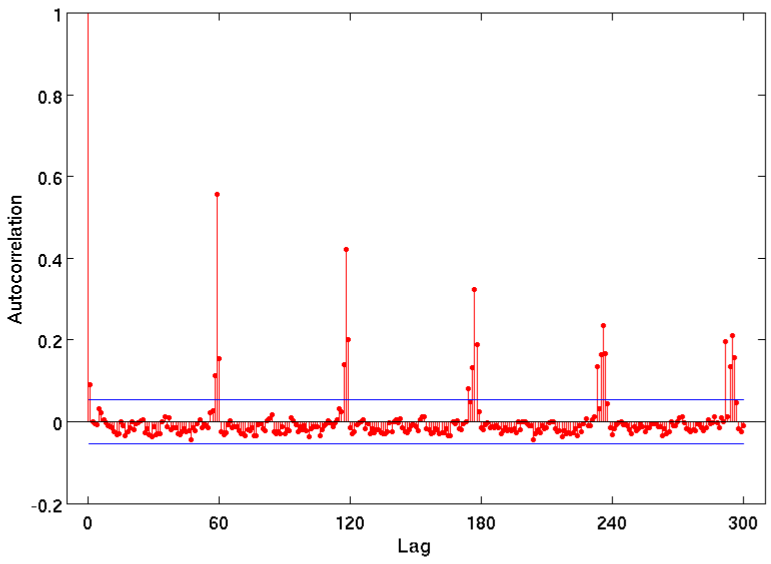


Figure 3.5: Autocorrelation function for $\overline{T}_{\mathrm{RET}}$

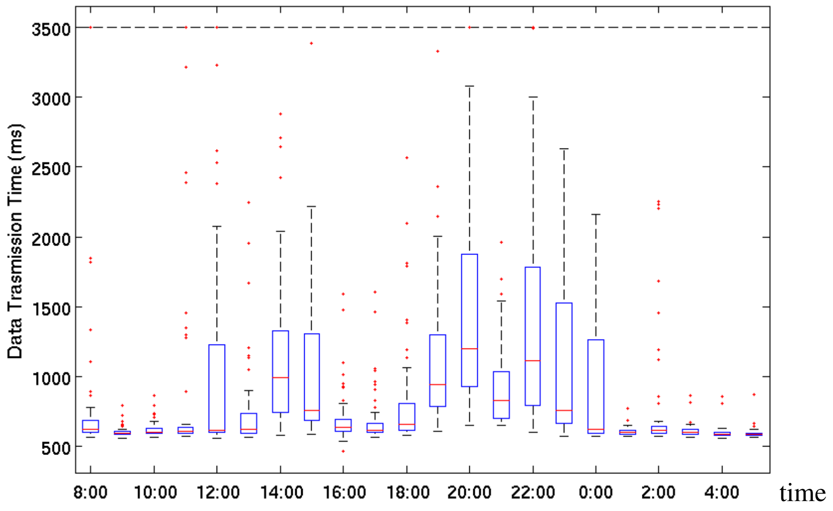


Figure 3.6: Box-plot of the data transmission time over unreliable network

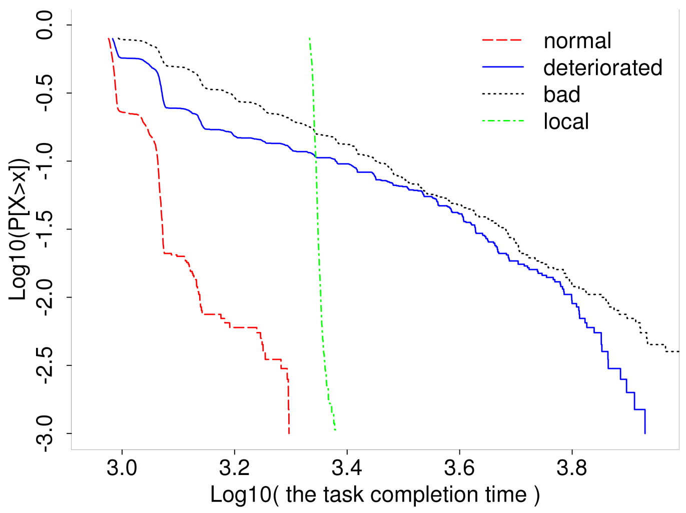


Figure 3.7: Log-log complementary distribution of the completion time

since enough outliers skew the distribution and increase the sample variance, the variability in the data is high enough to justify the use of restart.

3.2.1 Heavy-tail Distribution of Task Completion Time

It can be shown [142] that restart is beneficial if the task completion time follows a distribution with sufficiently high variance or heavy-tail. Therefore, in this section the sampled data will be analysed to determine whether the theoretical conditions for benefiting from restart are met. the distribution of the experimental data and its variability will be determined. The log-log complementary distribution plot is used to illustrate the weight of the tail of the distribution [38].

Fig. 3.7 shows the completion time of the three subsets and the local completion time versus their complementary cumulative distributions on a log scale. Clearly, for the subset of the bad network state the curve has an approximately constant slope of -2, indicating a heavy tail [38]. For the subset in deteriorated condition the tail has an exponential decay for long task completion times. Therefore in this case we cannot clearly diagnose a heavy-tailed distribution. For the normal subset the decrease is steep, for local computation completion times it is almost infinite. This indicates certainly no heavy tail in the latter two subsets.

Completion times using the local computation are almost constant. There is very little variation in the measurements. This means that once local computation has started restart will certainly not be beneficial. However, during a phase of poor network quality, a local restart may speed up the solution. This does not yet answer the question what a good choice of the timeout for restart could be.

Fig. 3.8-3.10 show the histograms of the three subsets and the density of the fitted distribu-

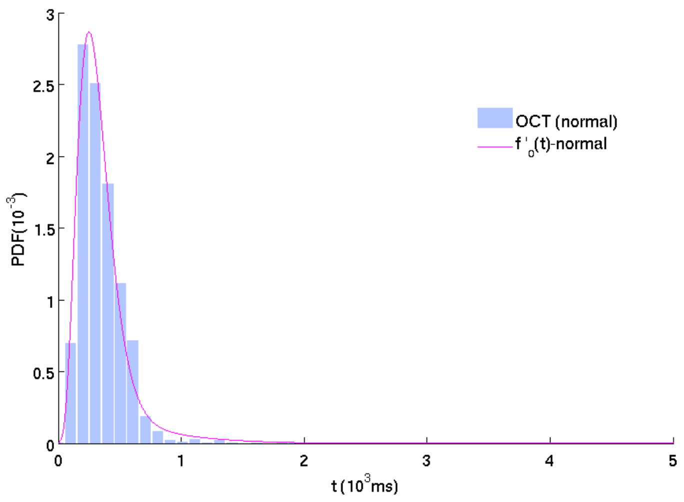


Figure 3.8: Histogram and PH distribution of the normal subset

tion. For convenient fitting of phase-type distributions the histograms have been shifted to the origin by subtracting the minimum value from all observations. The distribution fitting will be discussed in the next section.

3.2.2 Distribution Fitting

In this section we will describe the fitting process for the offloading completion time (OCT) as shown in the histograms and densities in Figs. 3.8-3.10. Let the random variable T_o represent OCT of an offloading task without restart. The distribution of T_o is fitted with the Cluster-based fitting algorithm [115] that fits a phase-type (PH) distribution to the data. The fitting procedure uses clustering and fits an Erlang distribution to each cluster. The full distribution is then a mix of those Erlang distributions, a hyper-Erlang distribution.

The hyper-Erlang distribution is suitable for situations where restarts succeed [119]. This distribution takes values from different random variables with different probabilities, for instance, with probability α_i a value from an Erlang distribution with m_i phases and parameter $\lambda_i > 0$, $i = 1, 2, ..., M$. M is the number of clusters. In general, the mixed-Erlang distribution is represented by a vector-matrix tuple ($\boldsymbol{\alpha}$, $\mathbf{Q}$).

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_1 & \mathbf{0} & & \\ & \ddots & \ddots & \\ & & & \mathbf{0} \\ & & & \boldsymbol{Q}_M \end{bmatrix}, \boldsymbol{Q}_i = \begin{bmatrix} -\lambda_i & \lambda_i & & \\ & \ddots & \ddots & \\ & & -\lambda_i & \lambda_i \\ & & & -\lambda_i \end{bmatrix} \tag{3.2}$$

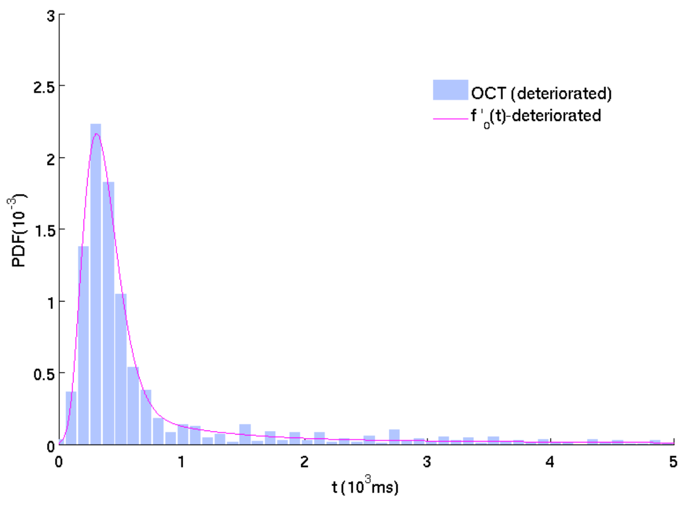


Figure 3.9: Histogram and PH distribution of the deteriorated subset

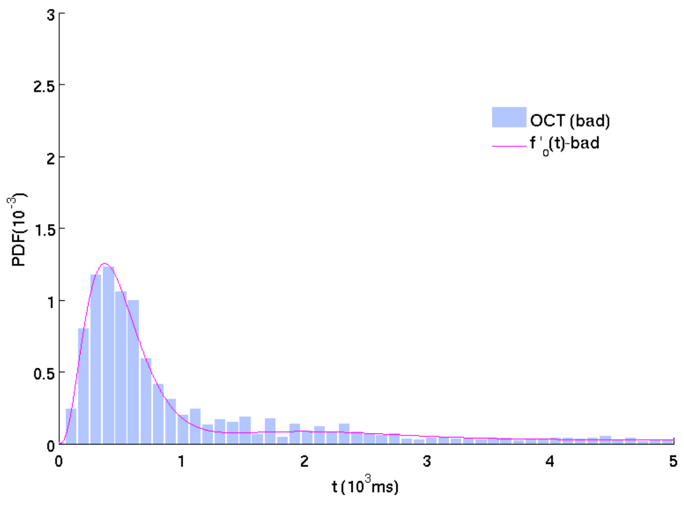


Figure 3.10: Histogram and PH distribution of the bad subset

$$\boldsymbol{\alpha} = (\underbrace{\alpha_1, 0, ..., 0}_{m_1}, \alpha_2, 0, ..., \underbrace{\alpha_M, 0, ..., 0}_{m_M},) \qquad \sum_{i=1}^{M} \alpha_i = 1 \tag{3.3}$$

$\boldsymbol{Q}_i \in \mathbb{R}^{m_i \times m_i}, i = 1, ..., M$ is a square matrix with size m_i. The probability density function and cumulative distribution function are defined as:

$$f(t) = \boldsymbol{\alpha} e^{\boldsymbol{Q}t}(-\boldsymbol{Q} \cdot \mathbf{I}) \tag{3.4}$$

$$F(t) = 1 - \boldsymbol{\alpha} e^{\boldsymbol{Q}t} \cdot \mathbf{I}, \tag{3.5}$$

where **I** is the column vector of ones with the appropriate size.

Although the hyper-Erlang distribution has exponentially decaying tails, its variance can still be large enough to fulfil the requirements for successful restart as formally introduced in Section 4.3.1. Since the completion times of a task have a lower threshold greater zero, as can be seen in Fig. 3.2 and PH-distributions preferably have a non-zero density at the origin, we have shifted the density $f_o(t)$ to the left by the minimum observed value T^o_{min} for T_o, i.e. $f_o(t) = f'_o(t - T^o_{min})$. This yields $f'_o(t)$ as the PH fitting result of the experimental data shifted to the origin.

Table 3.2: Hyper-Erlang parameters

	T^o_{min}	806	
		Phase-Type Distribution	
	m	λ	α
normal	[5, 2, 3]	[0.016, 0.0041, 0.0037]	[0.88, 0.047, 0.073]
deteriorated	[3, 6, 2]	[0.00082, 0.0163, 0.0023]	[0.1, 0.7, 0.2]
bad	[4, 8, 4]	[0.008, 0.0036, 0.001]	[0.7, 0.15, 0.15]

Fig. 3.8-3.10 show the histograms and the PH results of the shifted T_o of the normal, deteriorated and bad subset. We used three clusters to fit the data, $M = 3$. Since we grouped the data into three categories this seemed to be a natural choice as using 3 clusters for each subset. Of course, one could have chosen more clusters, which might have increased the goodness of fit. The parameter results are shown in Table 3.2.

Table 3.3: Error

	Normal	Deteriorated	Bad
$\triangle f$	0.2783	0.3051	0.2921
e_1	0.1077	0.0262	0.2894

Table 3.3 shows the error measurement of the PH results of the three subsets. We use the area difference between densities $\triangle f$ and the relative error in the first moment e_1 to measure the error. $\triangle f = \int_0^\infty |\hat{f}(t) - f(t)|dt$ and $e_1 = \frac{|\hat{c_1} - c_1|}{c_1}$, $f(t)$ denotes the empirical *pdf* of the distribution to be fitted, $\hat{f}(t)$ is the *pdf* of the PH result, c_1 and $\hat{c_1}$ is the first standardized moment of the empirical distribution and of the fitted PH distribution, respectively.

These distribution functions will be used later in the following chapters to verify that the condition of applying restart in mobile offloading systems is satisfied under the unstable network. More important, the optimal timeout to launch the restart is also mathematically derived from these distribution functions of task completion time.

3.3 Summary

In this chapter, we demonstrate that the network quality is not constant over time. By conducting experiment in our offloading test-bed, the variability of network quality in a local area network over one day is observed. When the network experiences a high traffic, the distribution of the task completion time in offloading is heavy-tailed. A long delay in data transmission is inevitable in this situation. In the next chapters, we will propose how to deal with this problem of unreliable network in mobile offloading with restart.

Part III

Failure Handling with Restart

Chapter 4

Restart Theory

All offloading systems mentioned so far may suffer under poor network condition and the application of well-designed fault-tolerance methods is in place. Restart is a simple and popular recovery scheme to mitigate network failures. It can be very effective for certain types of failures and its performance has been widely studied. Markov chain models and Laplace transforms have been developed to analyse the performance of restart for improving the expected task completion time [8, 85, 126, 86, 13, 32]. These analyses strongly support the efficiency of restart if the best restart timeout is known. Their implementation in an online algorithm for practical application is not straight forward. A fast method based on iteration theory to identify the optimal restart time is presented in [98]. The algorithm is improved in [141, 140, 142]. It is tailored for Internet applications in [117]. The authors of [118] introduce some performance metrics to measure the adaptivity of restart.

In the mobile offloading system, there is a selection between two types of restart: offloading retry or local retry. The two options give rise to multiple restart modes. First, the system can immediately restart local execution after the original offloading try has failed. Second, the local restart can be never used. The mobile device restarts with offloading infinitely until the task is completed. The last, a hybrid scheme coordinates the two types of restart. Prior to the final local restart, the system launches several offloading restarts. A threshold is configured to restrict the number of offloading restarts.

4.1 When Does Restart Work?

When using restart one has to decide whether and when to abort a running task and how to restart it. Obviously, there is a trade-off between waiting for the offloading task to complete and terminating the attempt to try again locally. In [142], an iterative solution for an infinite number of possible retries has been derived. In this section we derive an expression that formulates a condition under which restart in our offloading scenario will be beneficial.

The theoretical concept of restart applies to random variables for which, first, two successive tries are statistically independent and identically distributed, and, second, new tries abort previous attempts. In the mobile offloading system, the second assumption is certainly met. When the mobile device restarts the task by a local try it abandons the first try on the remote server, where it might continue to run, but will not influence further processing of the restarted task. However, the two successive tries are not drawn from the same distribution, as the computation time in the local device follows a different distribution than the offloading task. The offloading timeout might not be optimal, but completion of the task is guaranteed as the local computation always finishes.

For a given random variable T describing task completion time restart after a timeout τ is promising if the following condition holds [142]:

$$E[T] < E[T - \tau | T > \tau] \tag{4.1}$$

The interpretation of condition (4.1) means 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 [142] that condition (4.1) holds if the task completion time follows a distribution with sufficiently high variance or heavy-tail. In Section 3.2.1, we have shown that when the network quality deteriorates to a certain degree, the distribution of offloading task completion time has a heavy-tail. In the following sections the sampled experiment data shown in Chapter 3 will be analysed to determine whether the theoretical conditions for successful restart are met, and we also derive the optimal timeout after which to restart.

4.2 Single Local Restart

Obviously, when the offloading task fails, the mobile device may retry offloading or restart the task using the resources in the local mobile device instead of those in the Cloud. As introduced in [146], if the offloading task needs an unknown time to migrate computation through the unstable network connection, re-executing and completing the computations locally by the mobile device can save both time and energy. In this section, we adopt the solution for computing the optimal timeout from [142] for two tries and a single restart: a first attempt using offloading and a fall back local computation after expiry of the timeout.

4.2.1 Derivation of the Condition for a Single Local Restart

Remember that T_o represents the offloading completion time OCT of an offloading task without restart. Its density is $f_o(t)$ and its distribution function is $F_o(t)$. Assume τ is the restart time, at which the previous offloading task is aborted and the local computation is issued. Correspondingly, T_l represents the local computation time LCT of the same task, $f_l(t)$ its density and $F_l(t)$ its distribution. We assume that $F_o(t)$ and $F_l(t)$ are both continuous probability distribution functions defined over the domain $[0, \infty)$, such that $F_o(t) > 0$ and $F_l(t) > 0$ if $t > 0$. We introduce T to denote the completion time when a local restart is allowed. We write $f(t)$ and $F(t)$ for its density and cumulative distribution function, respectively. We are interested in the expectation of T using the optimal timeout τ for one local restart.

$$F(t) = \begin{cases} F_o(t) & (0 \leqslant t < \tau) \\ 1 - (1 - F_o(\tau))(1 - F_l(t - \tau)) & (\tau \leqslant t) \end{cases} \tag{4.2}$$

$$f(t) = \begin{cases} f_o(t) & (0 \leqslant t < \tau) \\ (1 - F_o(\tau))f_l(t - \tau) & (\tau \leqslant t) \end{cases} \tag{4.3}$$

Analogous to [142] we define the partial moments $M_n(\tau)$ of the completion time T to determine its expectation $E[T]$.

$$M_n(\tau) = \int_0^\tau t^n f(t)dt = \int_0^\tau t^n f_o(t)dt \tag{4.4}$$

The respective densities of T and T_o are identical between 0 and τ, so their partial moments are equal.

$$\begin{aligned} E[T^n] &= \int_0^\tau t^n f_o(t)dt + \int_\tau^\infty t^n(1 - F_o(\tau))f_l(t - \tau)dt \\ &= M_n(\tau) + (1 - F_o(\tau)) \sum_{k=0}^{n} \begin{pmatrix} n \\ k \end{pmatrix} \tau^{n-k} E[T_l^k] \end{aligned} \tag{4.5}$$

$$E[T] = M(\tau) + (1 - F_o(\tau))(\tau + E[T_l]) \tag{4.6}$$

A simple criterion to decide whether to restart or not can be formulated. If there exists an interval S in [0, ∞), where $\tau \in S \Rightarrow E[T] < E[T_o]$, then restart is beneficial. With (4.6), this condition can be written as the following inequality:

$$E[T_l] < \frac{\int_\tau^\infty t f_o(t)dt}{1 - F_o(\tau)} - \tau \tag{4.7}$$

Since the data has been shifted to the origin (4.7) has to be adjusted to

$$E[T_l] < \frac{\int_{\tau - T^o_{min}}^\infty t f'_o(t)dt}{1 - F'_o(\tau - T^o_{min})} - (\tau - T^o_{min}) \tag{4.8}$$

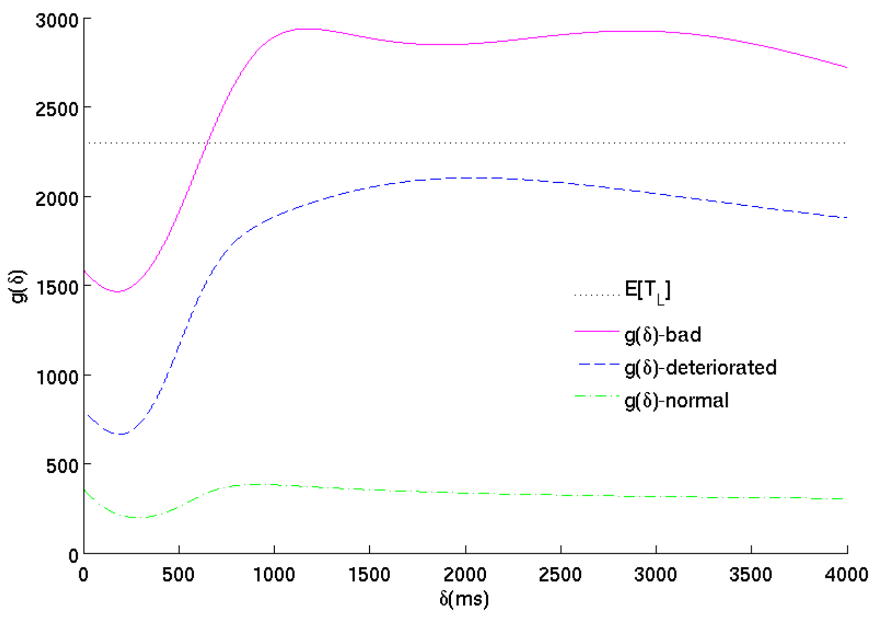


Figure 4.1: Restart timeout for the different subsets of the data

The optimal restart time is the value of τ where $E[T]$ is minimal. Hence, $f_o(t)$ is the key factor for finding the optimal τ and to take the decision to restart. As introduced in Section 3.2.1, $f_o(t)$ changes with the network quality. Accurately capturing $f_o(t)$ at run time gives a good solution, but it is a challenge. In Chapter 8, we will introduce a fast method to dynamically approximate $f_o(t)$. Before that, we use the previous experiment data to test the validity of the local restart condition (4.8).

4.2.2 Optimal Timeout for Single Local Restart

For convenience we use $g(\delta)$ to represent the right hand side of (4.8), $\delta = \tau - T^o_{min}$, i.e.

$$g(\delta) = \frac{\int_\delta^\infty t f'_o(t)dt}{1 - F'_o(\delta)} - \delta \tag{4.9}$$

The potential benefit of the local restart is expressed by $g(\delta)$ and $E[T_l]$ is the threshold to decide whether the local restart is useful or not. If the value of $g(\delta)$ is low, it indicates that the task has a high probability to be completed by offloading and local restart is not helpful. If the value of $g(\delta)$ is high, it indicates that the network condition is poor and the task completion has a high probability to be delayed. In this case restart can be very beneficial to the task.

Fig. 4.1 shows the result of (4.1), calculated according to $f'_o(t)$ of the three subsets from Table 3.2. $E[T_l]$ is calculated based on the data in Fig. 3.3. Only for values δ for which $g(\delta)$ is larger than the expected local completion time $E[T_l]$ a retry will be beneficial. It can be seen in Fig. 4.1

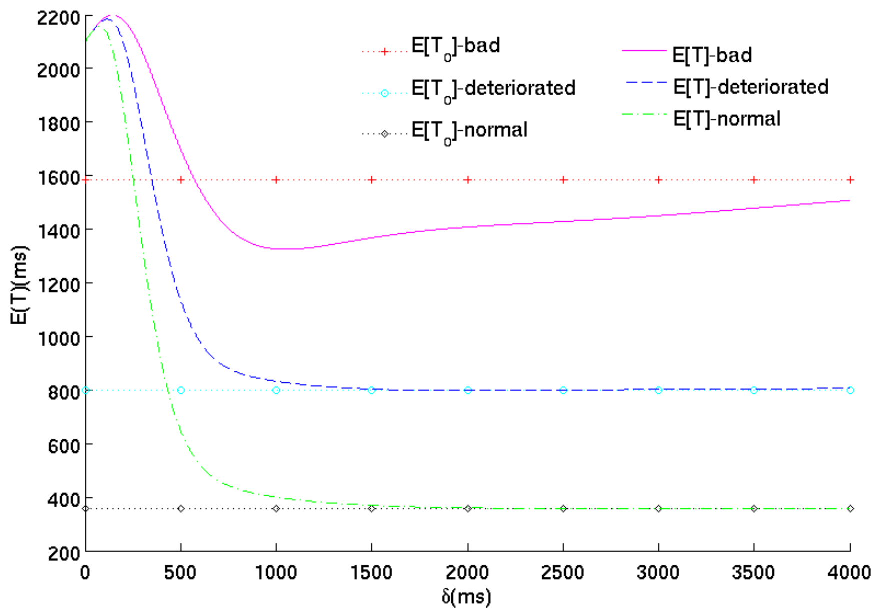


Figure 4.2: Expectation of OCT with/without the local restart versus τ

that such values only exist for the curve based on the bad subset of data.

However, Fig. 4.1 does not allow to determine the optimal restart timeout. We use the expectation of T as a metric to evaluate the system performance under different restart timeouts. The optimal time is found when $E[T]$ is minimal. Equation (4.6) is used to calculate $E[T]$.

For comparing the system performance with and without local restart, Fig. 4.2 shows $E[T]$ and $E[T_o]$ for the three subsets. As expected only $E[T]_{-bad}$ benefits from restart and even has a clear minimum under restart. The optimal restart time is found at the value for δ, for which $E[T]_{-bad}$ is minimal. We can confirm observations we already made earlier for restart, that when in doubt, one should rather set the timeout a litter larger. A too large timeout may not be optimal, but still better than too early retry. Because Fig. 4.2 confirms the observation that a too small restart timeout can be detrimental to the expected task completion time. The figure also shows that none of the other subsets benefit from restart.

Since network states often change, the histogram should be updated accordingly. The real time histogram is expected, hence a dynamical method is needed. In Chapter 8, we propose a fast and simple method to dynamically update the histogram and to estimate the restart condition directly from the histogram without first fitting a distribution.

4.3 Optimal Adaptive Restart

In this section, we adopt a similar method as in the last section to derive an expression for the expected task completion time of a hybrid adaptive restart scheme. The hybrid restart launches several offloading restarts before the final local restart. A threshold is configured to restrict the number of offloading restarts. Based on the theoretical analysis, the optimal timeout for every restart and the optimal threshold for the number of offloading restarts are derived. The sampled data used to demonstrate the theoretical analysis results is collected from experiments. Details about the data collection and the distribution fitting process have been introduced in Chapter 3.2.

4.3.1 Derivation of the Condition for Adaptive Restart

Obviously, in the mobile offloading system, the prerequisite of adopting offloading instead of local execution is:

$$E[T_o] < E[T_l] \tag{4.10}$$

If the mean completion time when offloading the task is longer than that of local execution, the offloading loses its advantage and should be rejected.

We still use T to denote the completion time when restart is allowed. We are interested in the expectation of T using the optimal timeout τ. The value of τ can be changed at real time according to system performance. But for simplifying the theoretical analysis, we assume that τ is constant in the process of completing a given task. At first, we take the same value of τ for both local and offloading restart. Then, in the next subsection, we use individual timeout values for the restart with offloading and local execution.

$$F(t) = \begin{cases} F_o(t) & 0 \leqslant t < \tau \\ 1-(1-F_o(\tau))(1-F_o(t-\tau)) & \tau \leqslant t < 2\tau \\ \quad\vdots & \\ 1-(1-F_o(\tau))^{n-1}(1-F_o(t-(n-1)\tau)) & \\ & (n-1)\tau \leqslant t < n\tau \\ 1-(1-F_o(\tau))^{n}(1-F_l(t-n\tau)) & n\tau \leqslant t \end{cases} \tag{4.11}$$

$$f(t) = \begin{cases} f_o(t) & 0 \leqslant t < \tau \\ (1-F_o(\tau))f_o(t-\tau) & \tau \leqslant t < 2\tau \\ \quad\vdots & \\ (1-F_o(\tau))^{n-1}f_o(t-(n-1)\tau) & \\ & (n-1)\tau \leqslant t < n\tau \\ (1-F_o(\tau))^{n}f_l(t-n\tau) & n\tau \leqslant t \end{cases} \tag{4.12}$$

Remember in equation (4.11) and (4.12), that $n \geqslant 2$, when $n = 1$ is the single restart mode which has been analysed in Section 4.2 as:

$$F(t) = \begin{cases} F_o(t) & (0 \leqslant t < \tau) \\ 1 - (1 - F_o(\tau))(1 - F_l(t - \tau)) & (\tau \leqslant t) \end{cases} \tag{4.13}$$

$$f(t) = \begin{cases} f_o(t) & (0 \leqslant t < \tau) \\ (1 - F_o(\tau)) f_l(t - \tau) & (\tau \leqslant t) \end{cases} \tag{4.14}$$

We define the partial expectation $M(\tau)$ of the completion time T to determine its expectation $E[T]$.

$$M(\tau) = \int_0^\tau t f(t) dt = \int_0^\tau t f_o(t) dt \tag{4.15}$$

The respective densities of T and T_o are identical between 0 and τ, so their partial expectations are equal as well.

$$\begin{aligned} E[T] = & \int_0^\tau t f_o(t) dt + (1 - F_o(\tau)) \int_\tau^{2\tau} t f_o(t - \tau) dt + \cdots \\ & + (1 - F_o(\tau))^{n-1} \int_{(n-1)\tau}^{n\tau} t f_o(t - (n-1)\tau) dt \\ & + (1 - F_o(\tau))^n \int_{n\tau}^{\infty} t f_l(t - n\tau) dt \\ = & M(\tau) + (1 - F_o)(\tau)(M(\tau) + \tau F_o(\tau)) + \cdots \\ & + (1 - F_o(\tau))^{n-1}(M(\tau) + (n-1)\tau F_o(\tau)) \\ & + (1 - F_o(\tau))^n (n\tau + E[T_l]) \\ = & \sum_{k=0}^{n-1} (1 - F_o(\tau))^k (M(\tau) + k\tau F_o(\tau)) \\ & + (1 - F_o(\tau))^n (n\tau + E[T_l]), \end{aligned} \tag{4.16}$$

$$\begin{aligned} & (1 - F_o(\tau))^n n\tau + (1 - F_o(\tau))^{n-1}(n-1)\tau F_o(\tau) \\ = & (1 - F_o(\tau))^n \tau + (1 - F_o(\tau))^n (n-1)\tau \\ & + (1 - F_o(\tau))^{n-1} \tau F_o(\tau) \\ = & (1 - F_o(\tau))^n \tau + (1 - F_o(\tau))^{n-1}(n-1)\tau, \end{aligned} \tag{4.17}$$

$$\begin{aligned}&\sum_{k=0}^{n-1}(1-F_o(\tau))^k k\tau F_o(\tau)+(1-F_o(\tau))^n\tau\\ =&\tau\sum_{k=1}^{n}(1-F_o(\tau))^k.\end{aligned} \tag{4.18}$$

Substituting (4.18) in equation (4.16), we get

$$\begin{aligned}E[T]=&M(\tau)+\sum_{k=1}^{n-1}(1-F_o(\tau))^k(M(\tau)+\tau)\\ &+(1-F_o(\tau))^n(\tau+E[T_l]).\end{aligned} \tag{4.19}$$

Let $E[T_i]$ represent $E[T]$ when $n=i$, and $E[T_\tau]$ means $n\to\infty$. When $n\to\infty$ is the second restart mode with infinite offloading restart, as derived in [142]:

$$E[T_\tau]=\frac{M(\tau)+\tau}{F_o(\tau)}-\tau. \tag{4.20}$$

When $n=1$,

$$E[T_1]=M(\tau)+(1-F_o(\tau))(\tau+E[T_l]). \tag{4.21}$$

The criterion to decide whether to restart or not can be formulated as $E[T]<E[T_o]$. With (4.19), this condition can be written as the following inequality:

$$E[T_l]<\frac{\int_\tau^\infty tf_o(t)dt}{(1-F_o(\tau))^n}-\sum_{k=1}^{n-1}\frac{M(\tau)+\tau}{(1-F_o(\tau))^k}-\tau \tag{4.22}$$

In [150] and the last section, the restart criterion for $n=1$ has been derived as

$$E[T_l]<\frac{\int_\tau^\infty tf_o(t)dt}{1-F_o(\tau)}-\tau \tag{4.23}$$

For $n\to\infty$, the restart criterion is

$$\frac{M(\tau)+\tau}{F_o(\tau)}-\tau<E[T_o]. \tag{4.24}$$

Theorem 1. When the criterion of exclusive local restart ($n=1$) is satisfied, the criterion of the adaptive restart scheme ($n>1$ or $n\to\infty$) is also satisfied.

Proof.

As the prerequisite of using offloading is $E[T_o]<E[T_l]$, equation (4.23) can be extended as:

$$E[T_o] < E[T_l] < \frac{E[T_o] - M(\tau)}{1 - F_o(\tau)} - \tau \tag{4.25}$$

$\Longrightarrow$

$$E[T_o](1 - F_o(\tau)) < E[T_o] - M(\tau) - \tau(1 - F_o(\tau))$$

$\Longrightarrow$

$$M(\tau) + \tau(1 - F_o(\tau)) < E[T_o]F_o(\tau)$$

$\Longrightarrow$

$$\frac{M(\tau) + \tau}{F_o(\tau)} - \tau < E[T_o].$$

So when $n \to \infty$, **Theorem 1.** is true, let

$$g_1(\tau) = \frac{\int_\tau^\infty t f_o(t)dt}{1 - F_o(\tau)} - \tau. \tag{4.26}$$

When $n \geqslant 2$,

$$g_n(\tau) = \frac{\int_\tau^\infty t f_o(t)dt}{(1 - F_o(\tau))^n} - \sum_{k=1}^{n-1} \frac{M(\tau) + \tau}{(1 - F_o(\tau))^k} - \tau. \tag{4.27}$$

Remember the result of (4.24), we have

$$\begin{aligned}
&g_n(\tau) - g_{n-1}(\tau) \\
=&\frac{\int_\tau^\infty t f_o(t)dt}{(1 - F_o(\tau))^{n-1}}(\frac{1}{1 - F_o(\tau)} - 1) - \frac{M(\tau) + \tau}{(1 - F_o(\tau))^{n-1}} \\
=&\frac{E[T_o] - M(\tau)}{(1 - F_o(\tau))^n}F_o(\tau) - \frac{M(\tau) + \tau}{(1 - F_o(\tau))^{n-1}} \\
>&\frac{\frac{M(\tau) + \tau}{F_o(\tau)} - \tau - M(\tau)}{(1 - F_o(\tau))^n}F_o(\tau) - \frac{M(\tau) + \tau}{(1 - F_o(\tau))^{n-1}} = 0.
\end{aligned} \tag{4.28}$$

We have proved that $g_n(\tau) > g_{n-1}(\tau)$, when $n \geqslant 2$. Thus if $g_1(\tau) > E[T_l]$, the inequality of (4.22) is always true for any $n > 1$, including $n \to \infty$.

Fig.4.3 shows the result of (4.27), calculated according to $f_o(t)$ of a more representative set of data introduced in Appendix A, Table A.1. **Theorem 1.** is also demonstrated in Fig.4.3. An interesting point worth to mention is that when $(M(\tau) + \tau)/F_o(\tau) - \tau = E[T_o]$, τ is at the critical value and $g_n(\tau) = g_{n-1}(\tau) = \cdots = g_1(\tau) = M(\tau)$. When τ is larger than this critical value, $g_n(\tau)$ becomes an increasing function with n at the same τ. Thus if $g_1(\tau) > E[T_l]$, then $g_n(\tau) > E[T_l]$.

Theorem 2. When the criterion of restart is satisfied as the inequality (4.24) holds, the expected task completion time $E[T_n]$ decreases with the number of restart n at the same τ.

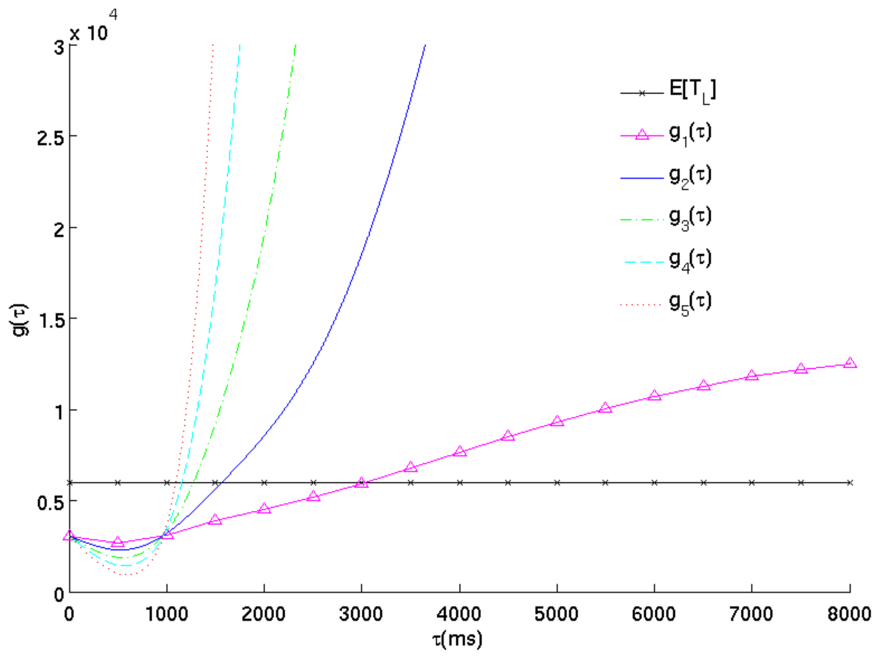


Figure 4.3: Restart timeout with 0∼4 offloading retries and 1 local retry

Proof.

Using the prerequisite of utilizing offloading, $E[T_o] < E[T_l]$, the inequality of (4.24) can be extended as:

$$\frac{M(\tau)+\tau}{F_o(\tau)} - \tau < E[T_l]. \tag{4.29}$$

When $n > 1$, we have $E[T_n] < E[T_{n-1}]$:

$$\begin{aligned} & E[T_n] - E[T_{n-1}] \\ =&(1-F_o(\tau))^{n-1}(\tau + M(\tau)) + (1-F_o(\tau))^{n-1}(\tau \\ & + E[T_l])(1 - F_o(\tau) - 1) \\ =&(1-F_o(\tau))^{n-1}((\tau + M(\tau)) - (\tau + E[T_l])F_o(\tau)) \\ <&(1-F_o(\tau))^{n-1}(\frac{(\tau+M(\tau))}{F_o(\tau)} - \tau - E[T_l]) = 0. \end{aligned} \tag{4.30}$$

When $n \to \infty$:

$$\begin{aligned} & E[T_\tau] - E[T_n] \\ =&\frac{M(\tau)+\tau}{F_o(\tau)} - \tau - M(\tau) - \sum_{k=1}^{n-1}(1-F_o(\tau))^k(M(\tau)+\tau) \\ & - (1-F_o(\tau))^n(\tau + E[T_l]), \end{aligned} \tag{4.31}$$

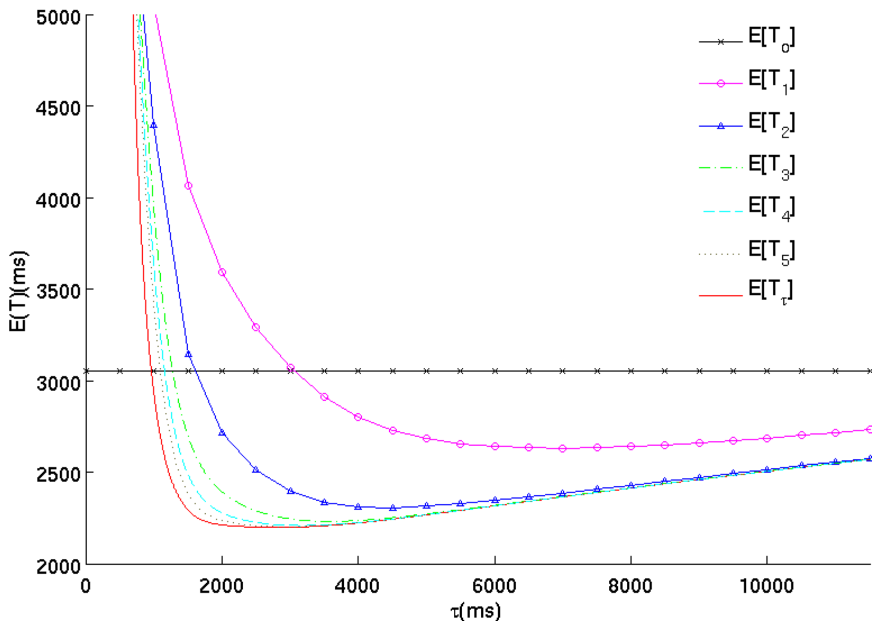


Figure 4.4: Expectation of the task completion time versus τ under different numbers of restarts

$$\sum_{k=1}^{n-1}(1-F_o(\tau))^k = \frac{(1-F_o(\tau))-(1-F_o(\tau))^n}{F_o(\tau)}, \tag{4.32}$$

$$\begin{aligned} &E[T_\tau]-E[T_n] \\ =&\frac{M(\tau)+\tau}{F_o(\tau)}-\tau-M(\tau)-\frac{1-F_o(\tau)}{F_o(\tau)}(M(\tau)+\tau) \\ &-\frac{(1-F_o(\tau))^n}{F_o(\tau)}(\tau+M(\tau))-(1-F_o(\tau))^n(\tau+E[T_l]) \\ =&(1-F_o(\tau))^n(\frac{M(\tau)+\tau}{F_o(\tau)}-\tau-E[T_l])<0. \end{aligned} \tag{4.33}$$

Thus if inequality (4.29) holds, we have $E[T_\tau] < E[T_n] < \cdots < E[T_1] < E[T_o] < E[T_l]$.

4.3.2 Identical Optimal Timeout

The optimal restart timeout is the value of τ where $E[T_i]$ is minimal. For comparing the system performance under the hybrid adaptive restart scheme with different number of restarts, Fig.4.4 shows $E[T_i](i = 1 \sim 5)$, $E[T_\tau]$ and $E[T_o]$ for the same sample data $f_o(t)$ of Table A.1. As expected $E[T_\tau]$ has the best performance, the minimum of $E[T_\tau]$ is the lowest. The most important observation shown in Fig.4.4 is that when $i = 2$, $E[T_2]$ is quite close to the ideal performance of $E[T_\tau]$. To evaluate the performance of different numbers of restarts, we define a metric to measure the distance between the optimal performance $E[T_\tau]$ and $E[T_i]$. As we have proved that $E[T_i] < E[T_o]$ for $i = 1, 2, \ldots, \infty$, the value of $E[T_i]$ can only vary in the space of $(E[T_\tau], E[T_o])$. So we define $d_i = (E[T_o] - E[T_i])/(E[T_o] - E[T_\tau])$ to measure how close $E[T_i]$ is to $E[T_\tau]$.

From Table 4.1, we can easily find that the performance of using two restarts (one local restart plus one offloading restart) can closely approach the best performance of infinite offloading restarts. Because $E[T_2]$ has reached 87% performance of $E[T_\tau]$, whereas if restarting exclusively with the local execution and no offloading restart, its performance $E[T_1]$ reaches merely 50% of $E[T_\tau]$. Accordingly, we can conclude that the adaptive restart scheme is better than the exclusive single local restart. This is because it uses the offloading restart to speed up the task completion. Although the performance of using more restarts is a little better, $E[T_4] < E[T_2]$, this result is in theory. In practice, the failure of the first offloading restart indicates a high possibility of the failure of the successive offloading restart. Actually, by applying local restart, the hybrid adaptive restart scheme can also avoid unpredictable waiting times of redundant offloading restart. We will use experiments to prove this observation in Chapter 8.

Table 4.1: Performance

	$E[T_1]$	$E[T_2]$	$E[T_3]$	$E[T_4]$	$E[T_\tau]$
Min	2630	2305	2230	2208	2198
τ	6901	4402	3576	3185	2818
d	0.49	0.87	0.96	0.98	1

4.3.3 Individual Restart Timeout

In the previous subsection, we have evaluated the performance of the adaptive restart scheme with identical optimal timeout value. In this subsection, applying a different τ_l as the timeout for the local restart is analysed. $T^{'}$ denotes the task completion time when τ_l is allowed.

$$F(t) = \begin{cases} F_o(t) & 0 \leqslant t < \tau \\ 1 - (1 - F_o(\tau))(1 - F_o(t - \tau)) & \tau \leqslant t < 2\tau \\ \quad\vdots & \\ 1 - (1 - F_o(\tau))^{n-1}(1 - F_o(t - (n-1)\tau)) & \\ & (n-1)\tau \leqslant t < (n-1)\tau + \tau_l \\ 1 - (1 - F_o(\tau))^{n-1}(1 - F_o(\tau_l)(1 - F_l(t - (n-1)\tau) - \tau_l) & \\ & (n-1)\tau + \tau_l \leqslant t \end{cases} \tag{4.34}$$

$$f(t) = \begin{cases} f_o(t) & 0 \leqslant t < \tau \\ (1-F_o(\tau))f_o(t-\tau) & \tau \leqslant t < 2\tau \\ \quad\vdots & \\ (1-F_o(\tau))^{n-1}f_o(t-(n-1)\tau) & \\ \qquad (n-1)\tau \leqslant t < (n-1)\tau + \tau_l & \\ (1-F_o(\tau))^{n-1}(1-F_o(\tau_l)f_l(t-(n-1)\tau)-\tau_l) & \\ \qquad (n-1)\tau + \tau_l \leqslant t & \end{cases} \tag{4.35}$$

As in the function (4.11) and (4.12), $n \geqslant 2$ in the above function (4.34) and (4.35). When $n = 1$, the situation is identical with (4.13) and (4.14), so we do not write them out again.

Using the similar expression as equation (4.16), we calculate the expectation of $T^{'}$ as

$$\begin{aligned} E[T^{'}] = & \sum_{k=0}^{n-1}(1-F_o(\tau))^k(M(\tau)+k\tau F_o(\tau)) \\ & +(1-F_o(\tau))^{n-1}(1-F_o(\tau_l))((n-1)\tau+\tau_l+E[T_l]) \end{aligned} \tag{4.36}$$

Unfortunately, it is impossible to give a simplified expression for $E[T^{'}]$ as for (4.19). Fig. 4.5 shows $E[T_2^{'}]$ for various values of τ with different τ_l. The bottom of the graph indicates the best timeout fraction (parameters in Table. 4.2). Comparing the two tables 4.1 and 4.2, we can find that using different timeout values for the offloading and local restarts individually can improve the performance. The minimum of $E[T_2^{'}]$ is less than that of $E[T_2]$. But considering the complexity of the computation for the optimal τ and τ_l, this performance increase is expensive. In addition, in real applications, it is difficult for a mobile device to estimate the optimal τ and τ_l online as it requires to simultaneously sort a two dimensional matrix. A large amount of time and energy is required for calculating the individual timeout values, but the benefit is limited. Thus, we cannot recommend the use of individual timeouts for the adaptive restart scheme.

Table 4.2: Performance

	Min	τ	τ_l	d
$E[T_2^{'}]$	2284	3767	6901	0.90
$E[T_3^{'}]$	2220	3266	6901	0.97

Before moving on to the next section of experiments, we briefly review the conclusions obtained in this section.

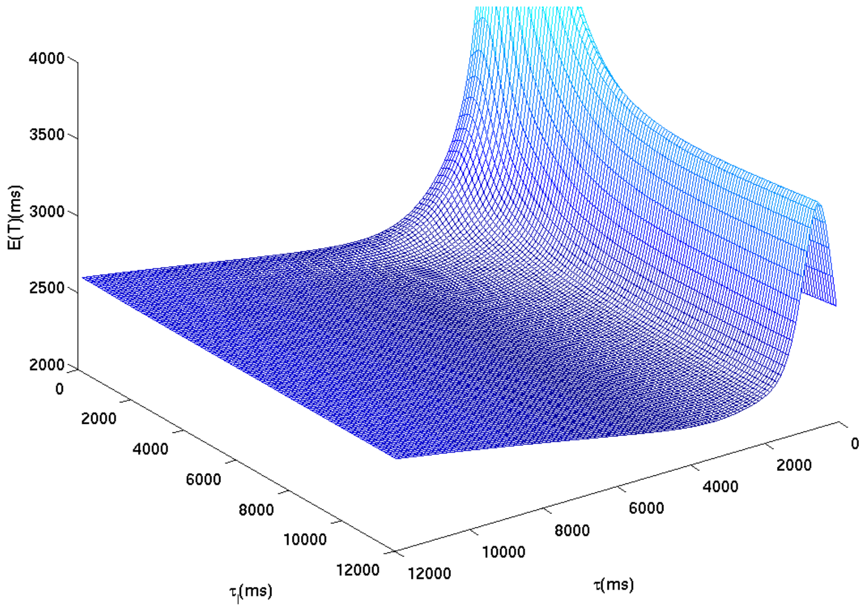


Figure 4.5: Expectation of the task completion time versus τ and τ_l

1 Theoretically, infinite offloading restart is the optimal option with the lowest mean completion time. But applying the adaptive restart scheme with two restarts (one offloading and one local) can reach almost 90% of the best performance.

2 Although triggering the offloading and local restart with individual timeout value can slightly reduce the expected completion time, due to its high complexity for calculating the optimal timeout values, we do not implement this scheme in the experiments.

4.4 Summary

In this Chapter, we have theoretically derived the condition for launching restart. From the analysis reault, we confirm that when the distribution of task completion time has high variance, restart is beneficial. We further derive the function to calculate the expected task completion time in the mobile offloading system when restart is enabled. The optimal restart timeout under three restart schemes are identified as the expected task completion time reaches its minimum. According to our analysis, applying different timeout values respectively for offloading restart and local restart cannot improve the system performance obviously, and an expensive overhead is required for this method. Thus, we propose using the identical and constant timeout value for every offloading or local restart.

Chapter 5

Offloading Program Engine

In section 2.1.3, we have introduced several existing mobile offloading frameworks. Although each of them has its individual merits, some common behaviours are shared by them. These behaviours are performed in both the mobile side and the server side. The execution and management of offloading tasks in the server involve much content concerning cloud computing, distributed computing and workload balancing. This is another important as well as interesting research filed which spans a wide range. In this thesis, we mainly focus on the mobile client, which can be monitored and analysed more easily. Using the mobile device as an entry point, some system behaviours can be observed.

Although the existing frameworks have provided a complete function of mobile offloading, their structures are mostly encapsulated like a black-box. They are not open-source and do not open any interface for public performance measurement. All data on performance of these frameworks are measured through inner channels which are reserved only for the developers. In order to understand the behaviour of the mobile offloading system facing the changing network quality and evaluate the performance of the restart scheme under different system configurations, we develop a full-functional mobile offloading engine ourselves. Through the reserved interface, we can measure most related data required for performance analysis.

In this chapter, we first describe an abstract work flow of the mobile client in the offloading system. Then we introduce our own mobile offloading engine. After that, we describe the restart process, which occurs when the mobile offloading system meets connection failures between the client and the server.

5.1 Work Flow of Mobile Client

We draw a general flow chart of the mobile device by using abstract state modules instead of the concrete functional components. The components constituting the mobile device are

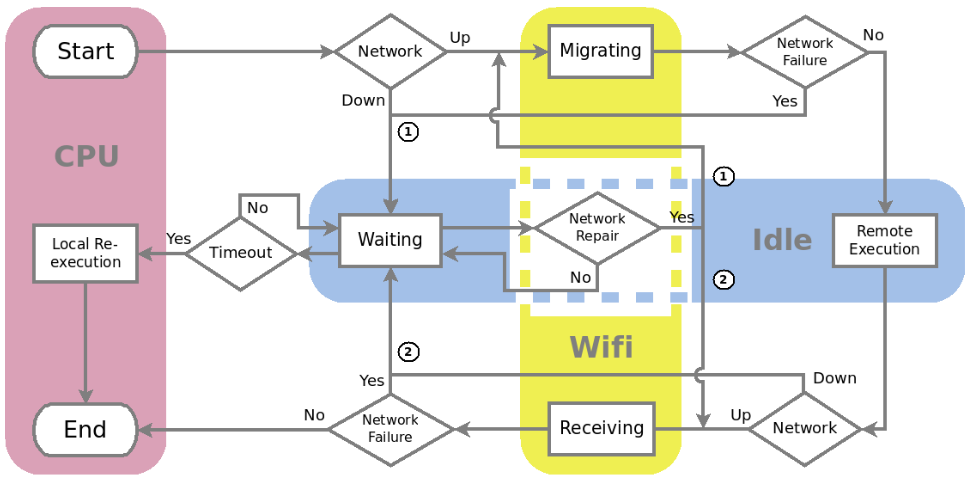


Figure 5.1: Flow Chart of the offloading process

categorised into the corresponding module according to their state in the execution of a mobile offloading task. Each of the modules are connected in the sequence of the working process of the mobile offloading system.

Referring to the most representative mobile offloading platform: MAUI, proposed in [39], we describe the offloading process as shown in the flow chat Fig.5.1. It has five states. The function of each state is listed as below:

- Migrating: The mobile device transmits the necessary information to the remote server for offloading tasks. The time spent in this state is related to the network condition and the data size. During this time, data transmission could be interrupted by the network failure. The mobile device has to restart this work after the network recovery.
- Remote Execution: After the mobile device completes the data transmission, offloaded tasks are executed in the remote server. The mobile device stays on in the current state, and waits for receiving the result from the remote server.
- Receiving: This state is the same as Migrating. When the network condition satisfies the offloading requirement, the mobile device receives the result of completed offload tasks from the remote server. If not, the mobile device goes into Waiting state.
- Waiting: If the wireless network fails, the offloading process moves to this state and the mobile device begins to count the waiting time. After the network was recovered, data transmission resumes again. But once the waiting time exceeds the timeout limit, the mobile device stops waiting and launches Local Re-execution.
- Local Re-execution: When the mobile device has waited a long time for network recovery, in order to avoid wasting more time, the pre-determined offloading task is locally executed by the mobile device instead of the remote server.

Both before and during the data transmission (Migrating and Receiving), the condition of the wireless network is monitored. When the network cannot support the data transmission, the mobile device moves to the state of Waiting. There are two inputs for Waiting (① and ②), they come from Migrating and Receiving individually. After the network recovery, the offloading process goes back to the state according to the input source. If the network is not stable, as the connection breaks frequently, the mobile device has to wait a long time for a sufficient up-time of the network to complete data transmission. In order to avoid the long waiting time, the task, which has decided to offload, is re-executed locally in the mobile device. It may need a longer execution time than offloading, but the execution continuity is maintained.

As we know, a mobile device consists of several components like CPU, Storage, Antenna, Battery and so on. During serving the function of each state in the offloading process, the utilizations of these components are different. In Migrating and Receiving, the utilization of antenna and other components related with wireless communication are high, but CPU is not heavyly loaded. Whereas in Local Re-execution, the utilization of CPU is close to 100%, an antenna is almost not used. Thus, when completing the function of a given state, we consider that these components make up a particular module. The power of this module equals the sum of every individual component power. As researched in [28], the energy consumed by each module does not change. Therefore, we classify the states based on the modules used by them. When focusing on theoretical analysis, for simplifying the calculation only three kinds of modules (CPU, WiFi and Idle) are used. It is assumed that the energy consumption of the states belong to the same module are equal. Migrating and Receiving are grouped into WiFi as they mainly use wireless components. Waiting and Remote Execution belong to Idle, because most components are in the idle state when the mobile device is waiting. In Local Re-execution, CPU of the mobile device undertakes many intensive computations, it consumes a lot of energy, thus it is seperated as a single category.

According to this workflow, a stochastic model is designed to analyse the operation of the mobile client. Details about the model are introduced in Chapter 6.

5.2 Engine Structure

For the purpose of closely seeing and dealing with the details of offloading, we create a modular and reusable offloading engine for applications with certain properties (a focused part with intensive computation). Our concept of mobile computation offloading is to have both the possibilities of executing a given application locally in a mobile device as well as remotely in a server. To do so, we copy the potentially offloadable application algorithm from the mobile device to the server. It is important to mention, that our engine will not provide a way to detect potentially offloadable parts of code, instead, we assume the programmer of each application has already

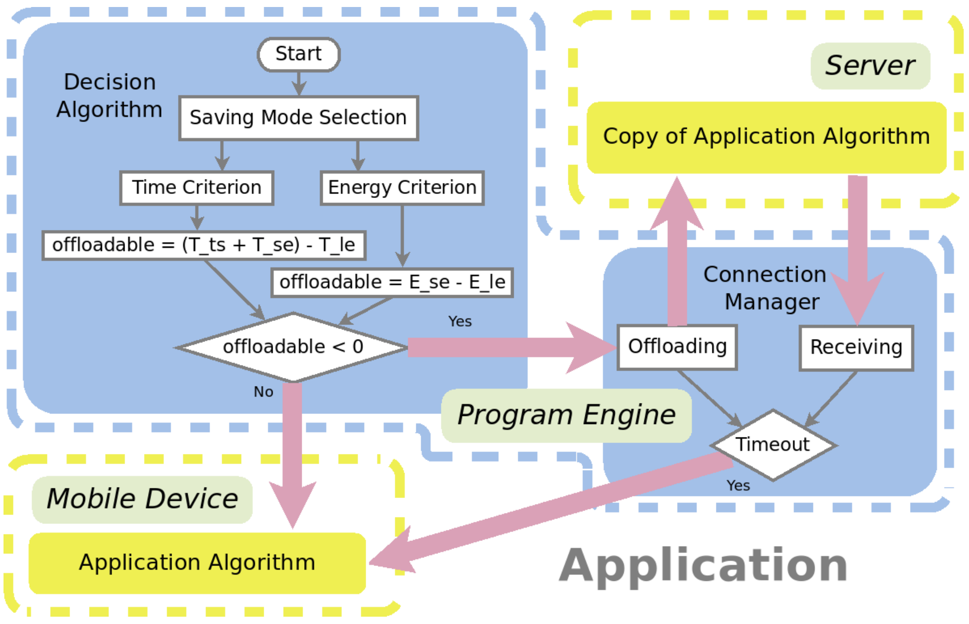


Figure 5.2: Structure of the application with the offloading engine

partitioned the application properly.

Fig. 5.2 shows a system architecture of an application equipped with the offloading engine. The given application algorithm is implemented both in the mobile device and the server. Our engine consists of two components: a decision algorithm and a connection manager. The decision algorithm decides whether to use offloading based on network conditions and user preferences. Once the engine decides to offload, the connection manager sends the input parameters that the application algorithm needs to the server through 3G, Wi-Fi or any kind of connection the mobile device has, and waits until the server answers with its result.

Our engine's decisions are based on the estimated computation and transmission costs. It is expected to provide a cost function for the application algorithms, which means that for a given input of the algorithm, this function is able to predict its cost. However, this is not easy, and only possible in the case of deterministic algorithms. They take the same time to calculate the output for a given input. For non-deterministic algorithms with random, pseudo-random or probabilistic execution flows, it will not be possible to estimate the cost. Thus, our engine estimates the cost based on past experience of using the same application algorithm. It is easy to understand that there is no need to offload if the application algorithm needs little time to be completed in the mobile device.

The input data sent by the mobile device are the cargo of offloading. We do not work with applications sending and receiving large amounts of data, for example, the case where the server could continuously answer with an audio or video stream is not considered. A timeout scheme is used to deal with connection failures during sending and receiving results.

5.2.1 Decision Algorithm

Considering the relevant parameters which affect the execution of offloading, we proposed the decision algorithm to decide whether it is worth or not to offload in a given situation. As the offloading aim is mainly to reduce the execution time and the battery consumption, two saving modes are optional in the engine: the *Time Criterion* and the *Energy Criterion*. The choice may depend on user preferences: there might be a user who prefers saving energy if the capacity of his battery is below 30%, while others might prefer saving execution time when they keep a battery with more than 50% capacity. Here the idea is not to combine both criteria in the decision step, but to provide automatic selection by assigning some weights to each part. For instance, the weight on the *Energy Criterion* will become larger while the level of battery capacity decreases. The decision algorithm will be run every time after finding a potentially offloadable part of code in an application that is using the offloading engine. The parameters in the flow chart of Fig.5.2 are introduced as below:

- T_ts: Data transmission time, which is the time spent by the data travels from the mobile device to the server and plus the time required by the server's answer comes back to the device again.
- T_se: Estimated server execution time, which is the time spent by the cloud server to complete the execution of the offloaded computation task.
- T_le: Estimated local execution time, which is the time spent for completing the execution of the offloaded computation task locally in the mobile device.
- E_se: Estimated offloading energy consumption, which is the energy consumed by the mobile device to send parameters and receive results with the cloud server.
- E_le: Estimated local execution energy consumption, which is the energy consumed by executing the offloaded task locally in the mobile device.

The values of all the parameters are collected and updated while running the given application. Before the first execution of an application using the engine, the initial attempt has to be made to obtain the necessary values of these parameters. Then the values of time parameters (T_ts, T_se and T_le) are going to be recalculated and updated in every execution. The data of energy parameters (E_se and E_le) are stored persistently, so they can be retrieved in future executions with no need to re-initiate them.

5.2.2 Connection Manager

After making the offloading decision, the program engine has to establish a connection with the cloud server to send parameters and receive results. While using this connection for data transmission, the condition of the wireless network is monitored. The connection manager is the module in charge of this work. It is also responsible for dealing with the connection failure by

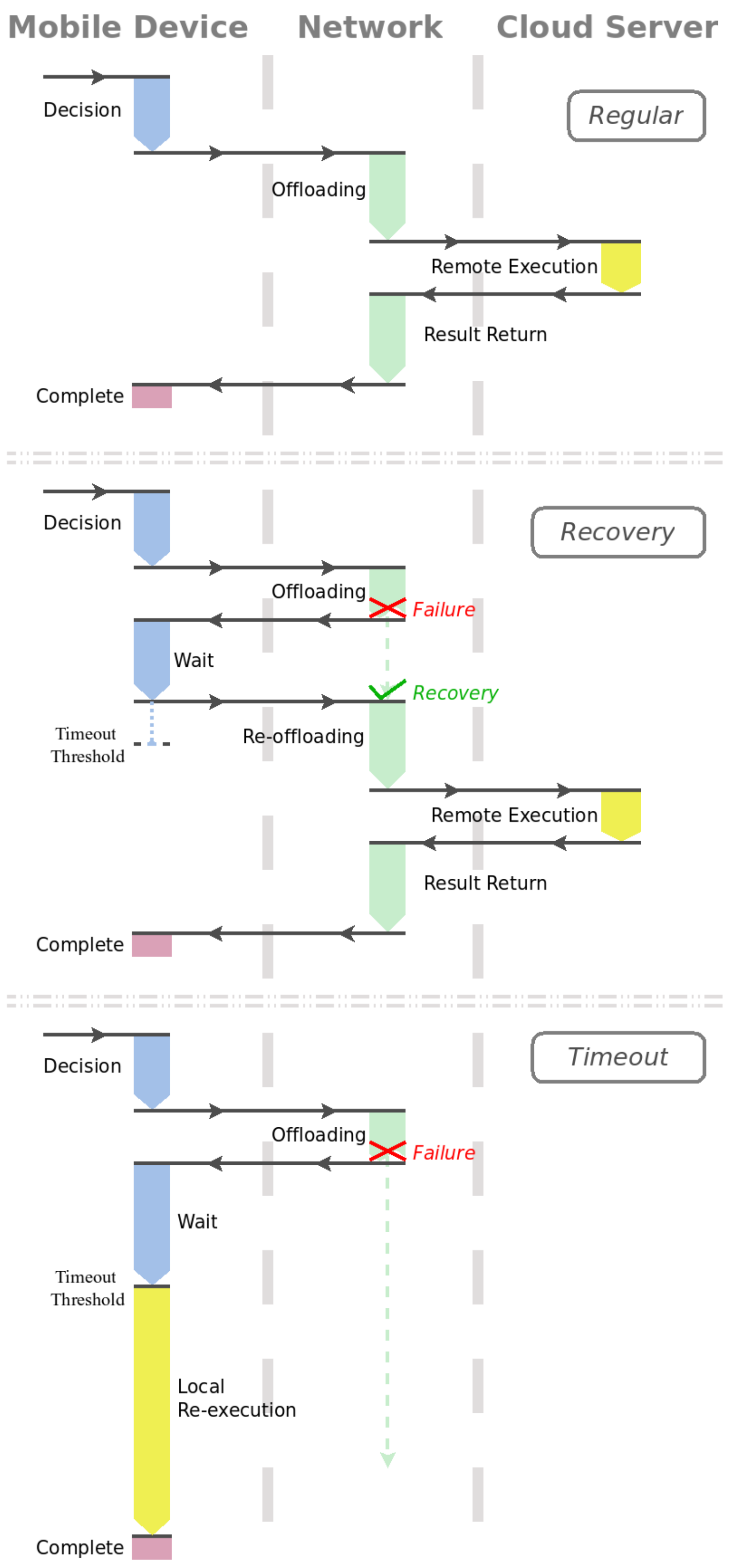


Figure 5.3: Process of the failure handling scheme in offloading

launching the local re-execution. Based on the proposal in [39], we describe the failure handling scheme as a kind of timeout restart method. The workflow of this timeout scheme under different situations is shown in Fig. 5.3.

In the offloading process, once the network fails and cannot support the data transmission, the connection manager informs the offloading engine to wait. A timeout value, or called threshold, is set to restrict the waiting time. If the network recovers quickly before the waiting time exceeds the threshold, the engine goes on to resume the execution of offloading. Generally, the offloaded computations are more time and energy intensive. Local re-execution may quickly use up the capacity of the battery in the mobile device. If the failed network can recover in a short time, as shown in Fig. 5.3, it is worth waiting for a moment rather than re-executing immediately. However, if the network is not robust enough, the connection may be lost frequently and be unable to recover in a short time. The mobile device has to wait a long time for a sufficiently long up-time of the connection to complete the data transmission. In order to avoid such a long waiting time, the pre-determined offloading task will be re-executed locally in the mobile device in case the waiting time exceeds the threshold.

5.2.3 Engine Implementation

The engine works with mobile devices using the Android system and the server supporting an ordinary web service. The mobile device calls the service by querying with simple HTTP requests. Apache Tomcat is implemented in the server, as it uses Java programming language, which is the same used by the Android applications. The server answers the queries in XML format, which is parsed in the mobile device to obtain the results of the offloaded computation tasks.

5.3 Restart Scheme

The objective of offloading is to reduce the task execution time by migrating heavy computations to the remote server. But the computation does not exclusively rely on the server. As mentioned in Section 5.2, our concept of mobile offloading is to provide both options, i.e. of executing a given application locally in a mobile device as well as remotely in a server and selecting the suitable one of the two. In the normal system state, the mobile device can establish a reliable connection with the server to send parameters and receive results. When the connection is interrupted or suffers degradation, the system is in some kind of a failed state. The failure handling consists of restart after expiry of a timeout. The workflow of normal offloading and adaptive restart is shown in Fig. 5.4.

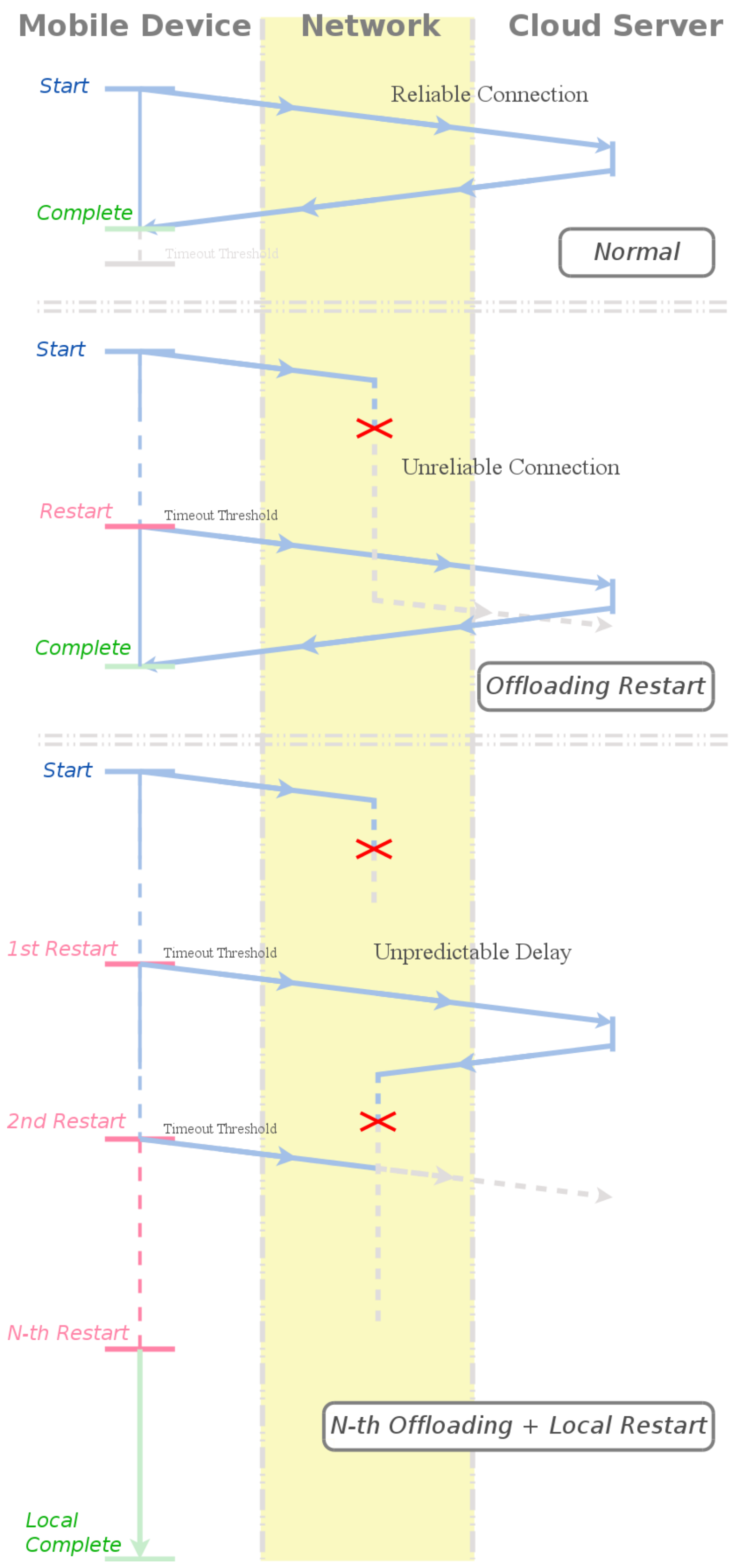


Figure 5.4: Process of the offloading execution with and without restart

Offloading Restart

We assume that during the offloading process, if the network connection is disturbed and cannot support the data transmission, the mobile device is informed to wait. A timeout value is set to restrict the waiting time. If the network recovers quickly before the timeout expires, the offloading process resumes execution. However, if the network is not robust enough, the connection is unable to recover in a short time. This is assumed to happen when the waiting time exceeds the timeout. In this case the previous task is abandoned, and a new try of the same offloading task is restarted.

Adaptive Restart

At times the offloading process may meet an unpredictable delay in data transmission. Various events can cause this delay, for example, one node on the path between the mobile device and the server can be overloaded. The offloading data can be blocked or even lost at this node. In this situation, repeated restart cannot solve the problem, on the contrary, it increases congestion. If the connection quality experiences a long term turbulence, the mobile device has to restart several times and waits long for a sufficiently long up-time of the connection to complete the data transmission. Obviously, the redundant restart consumes not only time but also too much energy. In order to avoid such a long waiting time, the offloading task will be restarted locally in the mobile device. Although the local restart may take longer than offloading, the execution continuity is maintained.

5.4 Summary

In this chapter, the structure of our mobile offloading engine is introduced. We implemented a decision algorithm into this engine to automatically migrate tasks to the server according to the user preference and the application property. The major difference of our engine with others is that our engine keeps the capability to complete the task locally in the mobile client. When the offloading task experiences a long delay caused by the unreliable network, restart is launched to increase the task completion. The work flows of several restart schemes are also illustrated in this chapter.

Chapter 6

Independent Static Model with Single Client

In this chapter, using Stochastic Activity Networks (SAN), many more system factors, like the processing rate of mobile devices, the delay of data transmission between clients and servers and the workload of offloading tasks will be considered. Besides the execution time, by referring to some experimental parameters, we derive the formulas to calculate the energy consumption.

We take two steps to evaluate the performance of local re-execution. At first, SAN is used as modelling technique. The measures are determined through simulation, and the parameters in the model are configured based on the experiment results of our program engine. Then, these measures are used to calculate the performance based on the metrics. The metrics are introduced in the second section.

6.1 Performance Analysis Models

Our models do not cover the whole system, they mainly focuse on the operation of the connection manager of the engine introduced in section 5.2. Other parts of the engine are simplified to reduce the simulation time. Two parts compose our model: a network model and an execution state model.

The network model is an independent model shown in Fig. 6.1. It is used to simulate the state changes of the wireless network during offloading. The execution state model in Fig. 6.2 is used to simulate the process of offloading in the mobile device. Each state in the offloading process is represented by the corresponding *marking* in our SAN models, and the execution of offloading is represented by *activities* which control the movements between those states. We also discuss the interactions between the two models and the factors which disturb the completion of offloading tasks.

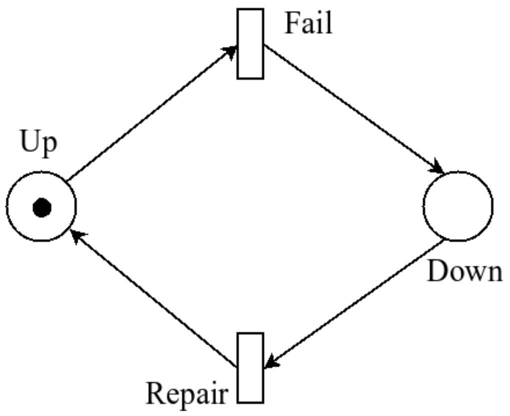


Figure 6.1: Wireless Network Model

6.1.1 Wireless Network Model

As can be seen in Fig. 6.1, this simple model only consists of two places *Up* and *Down*, two activities *Fail* and *Repair*. During transmission of data in the offloading process, it is easy to distinguish the two states of the wireless network. It is either up or down. If the network satisfies the offloading requirements, it is in the up state, parameters and results can be transmitted smoothly. Otherwise, the network is in the down state. Through the two activities *Fail* and *Repair*, the network condition turns from one state to the other after halting for a randomly distributed time, with the probability density function (pdf) $f(t)$. In this chapter, we use the exponential distribution as the $f(t)$ for *Fail* and *Repair*.

6.1.2 Execution State Model

Based on the process of offloading described before, the state movement in the execution state model is corresponding to the workflow of the connection manager of the engine. It is easy to find this relation by looking at Fig. 5.3 and Fig. 6.2. The execution state model consists of five places, the token departs from the initial place *Suspend*. Either through *Remote_Execution* or *Local_Re_execution*, it returns back. This denotes the end of an offloading task execution. The interpretations of the markings of the five places are as below:

- *Suspend*: The mobile device is preparing for offloading. When *Invoke* is activated, the token moves out and it indicates the offloading decision has been made. After the token returns back, it waits for a new offloading task to begin by activating *Invoke* again.
- *Migrating*: The marking of this place may denote two states: offloading and waiting. If the marking of *Up* is 0, the token in *Migrating* represents that the mobile device is waiting for network recovery. Otherwise, it represents that the mobile device is in the offloading state. Until *Data_Trans* is activated, the offloading step is successful. Before that, the mobile

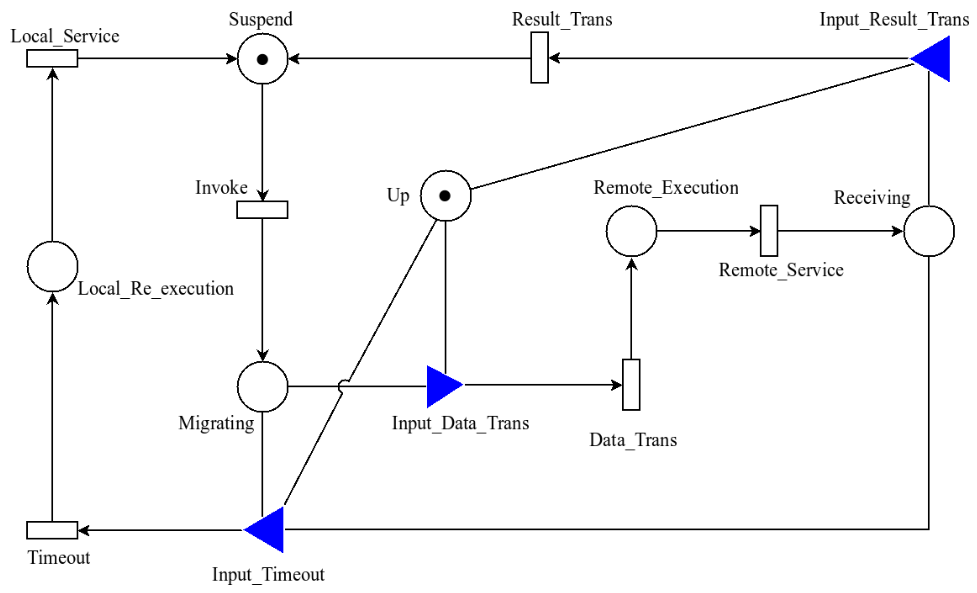


Figure 6.2: Execution State Model

device could move into the waiting state again, once the marking of *Up* changes from 1 to 0 as the enabling condition of *Data_Trans* is broken (Table 6.1). From this place, the engine starts the connection manager. The time spent in this place is related to the network condition.

- *Remote_Execution*: A token in this place denotes that the engine has successfully sent the parameters to the cloud server. Then it waits for the results returned by the could server after completing the offloaded computation.
- *Receiving*: This place is the same as *Migrating*. It denotes that the mobile device could be in one of the states receiving or waiting. If the network condition satisfies the requirement of returning results, the engine receives the completed results from the cloud server. If not, the mobile device goes into the waiting state. After *Result_Trans* is activated, the returning step succeeds.
- *Local_Re_execution*: This marking denotes that the connection manager has launched the local re-execution for the pre-determined offloading computation. Since the wireless connection failed, the waiting time has been counted by the engine. When it exceeds the threshold, in order to avoid wasting more time, the token moves to this place, which represents that the engine stops waiting and executes the offloading computation locally. After a local service time, modelled by activating *Local_Service*, the pre-determined offloading computation is completed in the mobile device.

This model shares the same place *Up* with the previous network model. Through the input gates, it controls the enabling condition of the three activities, *Data_Trans*, *Result_Trans* and

Table 6.1: Enabling predicate of the input gates

Gate	Predicate	Function
Input_Data_Trans	#*Up* $>$ 0 & #*Migrating* $>$ 0	#*Migrating* = 0; #*Remote_Execution* = 1;
Input_Result_Trans	#*Up* $>$ 0 & #*Receiving* $>$ 0	#*Receiving* = 0; #*Suspend* = 1;
Input_Timeout	(#*Up* == 0 & #*Migrating* $>$ 0), or (#*Up* == 0 & #*Receiving* $>$ 0),	#*Local_Re_execution* = 1; If(#*Migrating* $>$ 0) then #*Migrating* = 0; else if (#*Receiving* $>$ 0) then #*Receiving* = 0;

# refers to the number of tokens in the given place

Timeout, which change the marking of *Migrating* and *Receiving*. If the wireless network is not in the up state, the two activities *Data_Trans* and *Result_Trans* cannot be activated. The token only stays in *Migrating* or *Receiving* and waits for the activation of *Timeout*. When the wireless network returns to the up state, the two activities are enabled again. However, the activation of *Data_Trans (Result_Trans)* only moves the token from the place *Migrating (Receiving)* to the place *Remote_Execution (Suspend)*. The marking of *Up* is not controlled by this model, it only depends on the wireless network model.

The activity *Timeout* is used to control the threshold for launching local re-execution. In case the connection manager informs the engine to wait, the waiting time is recorded. If the network recovers before the threshold, the process of offloading is resumed as the enabling predicates of *Data_Trans* or *Result_Trans* are satisfied again. If the waiting time exceeds the threshold, *Timeout* is activated, which represents the local re-execution has been launched. Unlike the wireless network model, we apply the normal distribution to control the activation time. The mean values of normal distribution in the execution state model and their definitions are shown in Table 6.2. The activation time of *Timeout* is calculated using a fraction of the mean execution time of an offloading task. This offloading task includes the activations *Data/Result_Trans* and *Remote_Service*.

As we know, a mobile device consists of several components like CPU, storage, antenna, battery and so on. While serving different functions of the engine, the utilization of these components is different. In the connection manager, the utilization of the antenna and other components related with the wireless communication are high, but the CPU does not take too many jobs. Whereas in the local re-execution, the utilization of CPU is close to 100%, but the antenna is almost not used. Thus, when completing a given function of the engine, the entire energy consumption equals the sum of the energy used by every individual component. As found in

Table 6.2: Parameters for each activity (normal distribution)

Name	Parameter	Definition
Invoke:	$T_{arrival}$	The mean time before the next offloading request arrival.
Data/Result_Trans:	T_{trans}	The mean time to complete the data transmission.
Remote_Service:	T_{server}	The mean service time of offloaded tasks in remote servers.
Timeout:	$P \times (T_{server} + 2 \times T_{trans})$ *	*P* is the percentage of the entire execution time of an offloading task, it controls the threshold.
Local_Service:	T_{local}	The mean time to locally complete the offloading task in mobile devices.

* In the process, the system experiences two data transmission states: *Migrating* and *Receiving*. As the activation time of *Data_Trans* is the same as *Result_Trans*, the numerator is 2.

[28], the energy consumed by the component is related with the functionality. Therefore, we classify the functions into three categories (CPU, WiFi and idle) for simplifying the calculation.

It is assumed that the energy consumptions of the functions in the same category are equal. Sending and returning data are grouped into WiFi as they mainly use wireless components. Waiting and remote execution belong to idle, because most components are not running when the engine is in these states. In the local re-execution, the CPU of the mobile device undertakes many expensive computations, it consumes a lot of energy, thus it is separated as a single category.

6.2 Computing Performance

The model simulation only provides raw data. In order to compare the performance of different timeout values, the metrics are calculated from this data. In this section, three metrics are used to measure the performance: instability, energy consumption and throughput. We do not only analyse them independently. After normalizing the values of each metric, the three metrics are integrated through calculating their geometric distance from the best value in the same metric. The data of instability and throughput are directly received through simulation results. Energy consumption has to be calculated with the power parameters, which are measured through experiments.

6.2.1 Instability

The aim of this metric is to evaluate the delay caused by the network failure which interrupts the communication between mobile devices and cloud servers. The instability is defined as the probability that an offloading task experiences a connection failure while sending parameters or receiving results. In the model this is represented as the token halting in *Migrating* or *Receiving*, while the network is in the down state. As shown in Table 6.1, it is the same as the enabling predicate of *Input_Timeout*.

$$\begin{aligned} Pr_{instability} &= Pr((\#Migrating = 1 \vee \\ &\quad \#Receiving = 1) \wedge \#Up = 0) \end{aligned} \tag{6.1}$$

6.2.2 Throughput

The throughput *H*, which reflects the efficiency of our offloading engine, is always one of the most important metrics for any computer system. In this chapter, we defined *H* as the number of offloading tasks completed in the simulated system lifetime of a given simulation. In the model, it is represented as the number of *Invoke* activations.

6.2.3 Energy Consumption

In [28], Aaron Carroll made detailed measurements of the energy consumption of each module in a mobile device. Although their data is quite accurate, considering those expensive measurement instruments, we use the self-owned functions of Android SDK in our engine to measure and calculate the energy consumption. As introduced in [28], the energy consumption of executing different assignments are not equal. We define three power categories (p_{wifi}, p_{idle} and p_{cpu}) to distinguish them. Their values are also measured from the experimental data. The energy consumption of each stage in the offloading execution equals the holding time multiplied with the power. The holding time is how long the token stays in the corresponding place. The power of each stage is defined according to the category of its function [28], as introduced in the last section. The entire energy consumed in a given offloading process is:

$$E'_{re} = T_{trans} \times p_{wifi} + T_{idle} \times p_{idle} + T_{re} \times p_{cpu} \tag{6.2}$$

T_{trans} as in Table 6.2, is the time spent on offloading parameters and receiving results. It also includes the time wasted in the interrupted data transmission. T_{idle} consists of two parts T_1 and T_2, T_1 is the remote execution time. T_2 is the waiting time for the network recovery. As being restricted by the threshold, T_2 has an upper limit as $P \times (T_{server} + 2 * T_{trans})$. The local re-execution is launched when the waiting time exceeds the threshold. T_{re} is the time used for local re-execution, which has a positive correlation with the processing rate of the mobile device and the intensity of the offloaded computation. T_s is the total simulated system lifetime.

$$\begin{aligned} T_{trans} = Pr((\#Migrating = 1 \vee \\ \#Receiving = 1) \wedge \#Up = 1) \times T_s \end{aligned} \tag{6.3}$$

$$T_{idle} = T_1 + T_2 \tag{6.4}$$

$$T_1 = Pr(\#Remote\_Execution = 1) \times T_s \tag{6.5}$$

$$T_2 = Pr_{instability} \times T_s \tag{6.6}$$

$$T_{re} = Pr(\#Local\_Re\_execution = 1) \times T_s \tag{6.7}$$

Actually, the total energy consumption is related to the throughput *H* in the simulated system lifetime. To exclude this influence, the average energy consumption of each offload task E_{re} is calculated

$$E_{re} = E'_{re}/H. \tag{6.8}$$

6.2.4 Synthetical Performance Analysis

In order to comprehensively compare the performance of different timeout values, the three metrics are synthetically analysed. As the three metrics have quite different orders of magnitude and units, the normalization is used to transform all the data of the same metric into the range of [0,1]. The transform formula is:

$$y = \frac{x - Min(X)}{Max(X) - Min(X)} \tag{6.9}$$

After transforming, we found under the same timeout, the results of three metrics make up a three-dimensional vector $\langle u_i, e_i, h_i \rangle$ (u_i: instability, e_i: energy consumption, h_i: throughput). Thus, the performance of each timeout is defined as the geometric distance of this vector from the best one. For instability, it is expected to be the lower the better as for the energy consumption. But for throughput, if the system can complete more tasks in a given period, it has better performance. Therefore, the best vector is $\langle 0, 0, 1 \rangle$, and the distance is calculated as:

$$A = \sqrt{w_1 \times u_i^2 + w_2 \times e_i^2 + w_3 \times (1 - h_i)^2} \tag{6.10}$$

(w_1, w_2, w_3) is the weight vector for the three metrics. It can be adapted for different application scenarios. In this chapter, we only analyse the equal weight vector (1,1,1). Through the geometric distance, it provides a direct view of the performance comparison between different timeout values, which are controlled by the corresponding percentage.

6.3 Experiments and Simulations

In order to explore the effects of our connection failure handling scheme, which launches local re-execution based on a threshold, we take three steps. First, we design some test experiments to demonstrate the operation of our engine and collect the data of some parameters for our simulation model. Then, the model parameters are configured with the values, which are derived from the fitting result of the test experiment data. The system performance is calculated according to the formulas in section 6.2 and the simulation result. By comparing the performance, the optimal percentage for setting the timeout is found. Finally, we input the optimal timeout percentage into our engine, and verify its efficiency in a practical application scenario.

6.3.1 Test Experiment

In most situations, offloading is an efficient way to save both time and energy. Since not all applications are suitable for offloading, our engine uses the decision algorithm to separate the applications into offloadable and unoffloadable. For clearly illustrating the effect of the computation intensity on the offloading decision, a simple algorithm of iteration is applied as the sample application. It does nothing but counts up to a specific number. We test three mobile devices (Y: Samsung Galaxy Young, 832 MHz; N: Samsung Galaxy Nexus, 1.2 GHz Dual Core; S: Sony Xperia MT25, 1 GHz) to explain the decision algorithm of the Time Criterion. The same iteration algorithm is also implemented in a private server (4 cores: Intel Xeon CPU E5649 2.53 GHz) with Apache Tomcat 6, which is used as our cloud server. WiFi is used to support the connection between the mobile client and the server.

As shown in Fig. 6.3, the time spent for locally executing the iteration algorithm in the mobile device increases quickly with the number of iterations. But for remotely executing in the server, the execution time increases much slower. The reason is that, as the cloud server has a powerful computation capability, it can complete the iteration algorithm in a short time and it keeps a low rate of increase with the number of iterations. In addition, most of the remote execution time is used for the data transmission between the mobile device and the server. Since the data size of transmission almost remains the same, the transmission time does not change. The entire time of remote execution is mainly controlled by the number of iterations and it increases slowly.

In Fig. 6.3, the crossing point of the line "Local" and "Offload" is the watershed to divide the applications into offloadable or unoffloadable. Obviously, if the number of iterations is

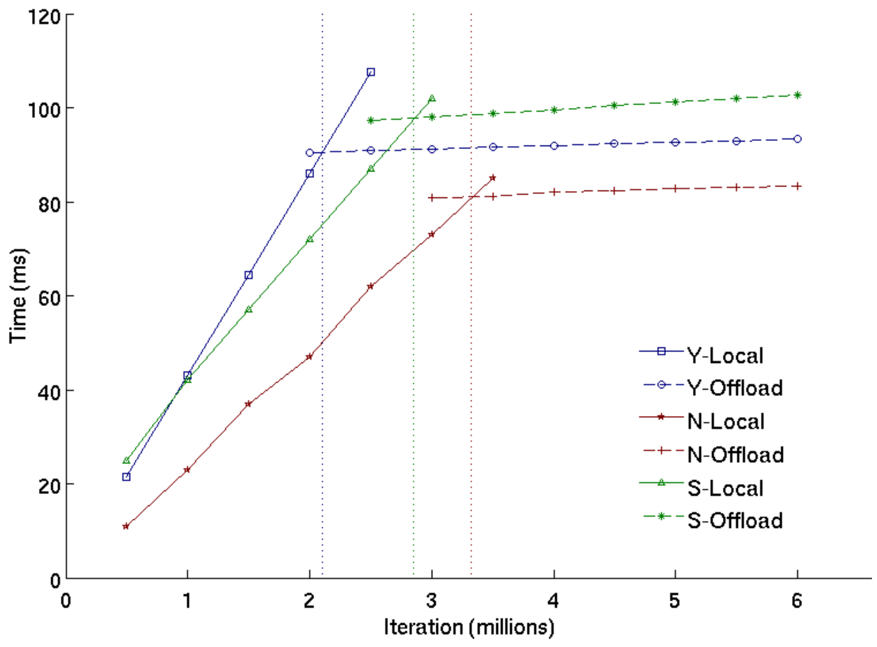


Figure 6.3: Execution Time of Iterations

larger than the vertical dotted line, the task is offloadable, if it is below this line, the task is unoffloadable. It can also be seen that a mobile device with a faster CPU completes the same intensive computation in a shorter time. However, for the remote execution time, the processing rate of the CPU is not the only factor. As shown in Fig. 6.3, the device “S” has a faster CPU than the device “Y”, but it needs more time to complete remote executions, because the device “Y” has a shorter data transmission time. Therefore, our engine takes the decision based on the practical execution data.

6.3.2 Parameter Fitting

After the test experiment, the data is fitted with a suitable distribution to obtain the model parameter values. Generally, Phase-type (PH) distributions are suitable to fit the time distributions in SAN models. But two factors make it improper for our models. The first one is the tremendous state space of the transition matrix. It needs more than a thousand phases to accurately fit our experiment data. This causes difficulties to configure this distribution in our model during simulation. The second one is the extended simulation time. Even applying only ten phases to configure the PH distribution, the simulation time is quite long.

Fig. 6.4 shows an example of the distribution fitting result. The histogram is the density function (*pdf*) of the execution time of running 1000 million of iterations in the mobile device. The sample space is 3000. We compare the fitting results of two distributions (Normal and PH). To fit with a PH distribution, we apply a cluster-based algorithm as proposed in [115],

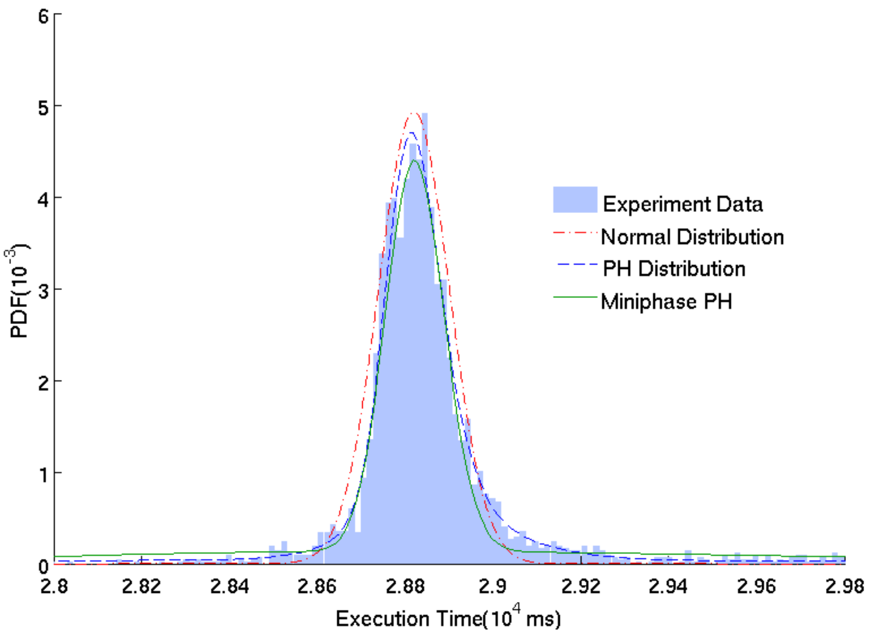


Figure 6.4: Execution Time Fitting with Distributions

which divides the samples into several clusters and individually fits each cluster with an Erlang distribution. Since the *pdf* has a long and heavy tail on both sides, fitting with one cluster cannot provide a good result for both the peak and the tail at the same time, at least using two clusters can make a satisfactory fitting result. The results are shown in Table 6.3. It is easy to find in the figure that the fitting result with more clusters and more phases is better than that with less clusters. "PH Distribution" in the figure is the closest to the histogram, especially in both transition parts close to the x-axis. "Miniphase PH" loses the accuracy in the transition parts and has a heavy left tail. We decided to use the normal distribution to simulate the stochastic time parameters in the execution state model. Although it is not as accurate as the PH distribution as shown in Fig. 6.4, it can save a lot of simulation time. And from Table 6.3, it can be seen that the transition matrix of PH distribution has more than 1000 phases. It is difficult to implement them in the model and it requires a long time for simulation. Therefore, we apply the normal distribution and the exponential distribution in our model for simulating.

6.3.3 Simulation

After fitting the experimental data, we obtain the values of our model parameters in a given application scenario. Then we simulate our model with Mobius [41]. Lagged Fibonacci random number generator is used with the seed of 31415. Mobius would need extended functionality, allowing to import a PH distribution parameter file and efficient random number generation to improve the situation. In the next chapter, we will introduce our method to implement PH

Table 6.3: Results of Distribution Fitting

Normal Distribution

Mean	Standard Deviation
28820	81

Phase-Type Distribution

N*	λ	α
1830	0.863	0.181
626	0.293	0.168
152	0.0702	0.183
2000	0.964	0.173
789	0.371	0.174
4	0.0015	0.119

PH Distribution with Minimized Phases

N*	λ	α
7	0.0029	0.288
1005	0.476	0.711

* Number of phases in each cluster.

distribution with Mobius. The simulated system lifetime of a simulation is fixed at 86400s (24 hours), which is long enough comparing with the execution time of a single offloading task. Performance of different timeouts under the same application scenario (Algorithm: 1000 million iterations, Mobile Device: Sony MT25, Server: Google Engine) are investigated by changing the percentage *P* in Table 6.2. The complete set of simulation parameters is provided in Table 6.4. We fixed the network failure and repair time only based on assumptions, they represent various network environments.

Since the performance is evaluated by its geometrical distance from the optimum, which is calculated using Eq. (10), a value close to zero is desirable. As shown in Fig.6.5, the performance comparison of various values of P with different T_{repair} is depicted. The bottom of the graph indicates the best timeout fraction (parameter P in Table 6.2) under different T_{repair}. We also show the performance of no re-execution in the end of the axis P. Apparently it is far from the minimum distance. Thus we could show that the local re-execution is effective to deal with network failure.

In Fig.6.6, we compare the optimal P under different network failure times. Each line in the figure shows the best P (bottom line in Fig. 6.5) with different network repair times under a fixed failure time. Apparently, the failure time has no effect on choosing the optimum, note those lines of different MTTFs are very close to each other. This behaviour can be justified by

Table 6.4: Model parameter values used in the simulation

parameter	List of values (in seconds)
Arrival rate $\lambda_{arrival}$	1/10
Local Execution Time T^*_{local}, V^*_{local}	28.820, 6.561
Data Transmission time T^*_{trans}, V^*_{trans}	0.168, 0.25
Remote service time $T^*_{server}, V^*_{server}$	3.502, 0.548
Mean times to failure T_{fail}	100, 500, 1000, 2000
Mean times to repair T_{repair}	60 values between 0.5 and 30
Percentage of timeout P^+	21 values between 0% and 200%

$^+$ refer to Table 6.2 item *Timeout*
* Mean and Variance of Normal distribution

an MTTF longer than the execution time of a single offloading task (Table 6.4). We assume that it is extremely unlikely to experience more than one failure in the same offloading process. Thus, once a failure happens, the repair time becomes the critical factor to determine the system performance. As shown in the figure, the optimal P increases with T_{repair}. It confirms our assumption that it is worth to wait a moment for the network recovery when meeting connection failures.

However, when T_{repair} becomes longer, the curves are not stable and suffer from severe oscillation. Referring to Fig. 6.5, it can be seen that the bottom field (dark blue area) becomes

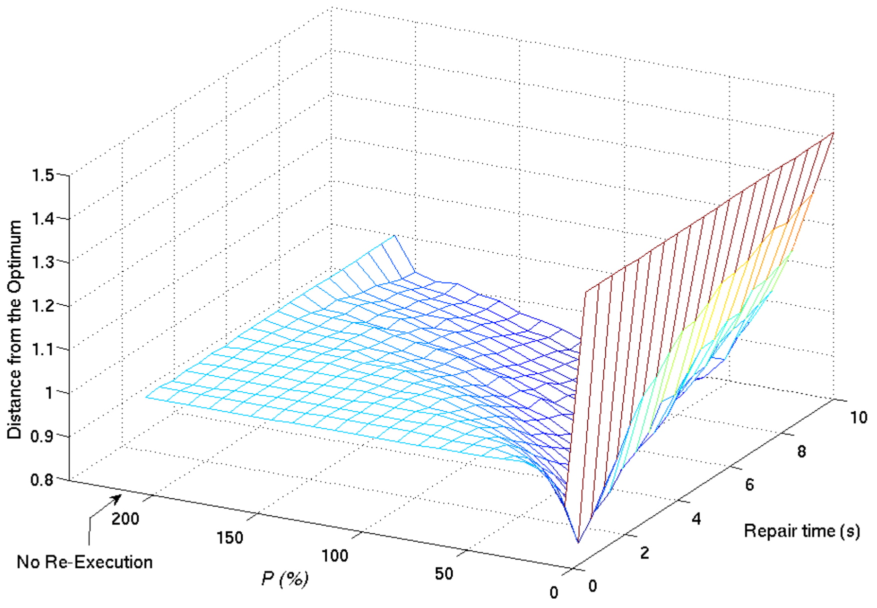


Figure 6.5: Performance Comparison with P and Network Repair Time

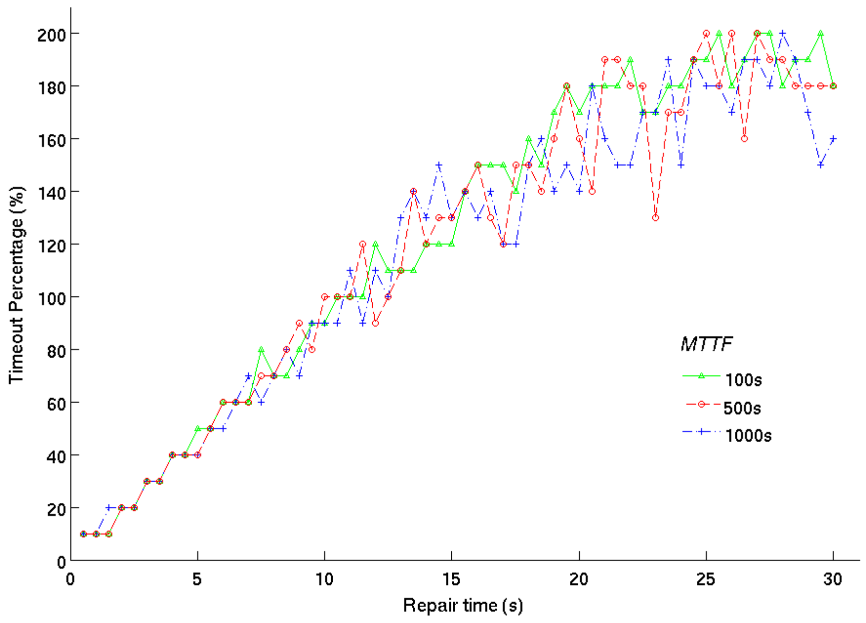


Figure 6.6: Optimal Timeout Value under different Failure and Repair Times.

smooth when T_{repair} is long. This makes it difficult to accurately determine the optimum. There are two solutions for this problem. The first one is extending the percentage to a wider range like 300%, and then comparing performance. Unfortunately, the result is still dissatisfying. The wave between $[150\%, 250\%]$ is stronger under the long T_{repair} ($> 20s$), and we are unable to find a stable optimum. Another solution is to comprehensively consider the performance of all T_{repair}. Actually in the real application scenario, it is impossible to predict the recovery time. So we try to find an optimal P^*, even if it cannot provide the best performance under all T_{repair}, it is still able to maintain an acceptable result for most T_{repair}. This P^* is viewed as the default optimum. For this target, we sum up all the distances of T_{repair} over P in Fig. 6.5. Then the default optimal timeout fraction can be found by comparing the sum of the distances of different fractions. In order to provide a clear view, we show the summed distances on a log-scale (base 10). As shown in Fig. 6.7, the shape of the transformed result is still like a valley and the bottom is the default optimal timeout P^*.

6.3.4 Verification Experiment

We use the optimal timeout fraction as input for our engine to verify its efficiency. The iteration algorithm (1000 millions) is used again as the sample application. The wireless network environment is in a campus which has been almost completely covered by WiFi. But there are still some blank areas without wireless signal. So when we take the mobile device and move around the campus, it experiences connection failure and recovery intermittently. We have tried to use PH distribution to simulate the failure or recovery time. However, in this case, the interval between failures is strongly stochastic. Without a tremendous number of tests, it is difficult to provide sufficient samples for fitting. So in simulation, we use the exponential distribution for MTTF and MTTR. The experiment results are shown in Fig. 6.7. To analyse the performance, we used two metrics: throughput and energy consumption, under a fixed experiment time of 30 minutes.

It is obvious that the throughput increases with the timeout P, and reaches the top when arriving at the optimum P^*. This indicates that, comparing with the long local re-execution time, waiting for the network recovery can save time and improve throughput because the offloading can complete the intensive computation more quickly. However it is restricted by an upper limit, which means there is no benefit of waiting indefinitely. Moreover, the energy consumption increases with the timeout P after passing the optimal percentage. It is easy to understand that when P is low, a large number of local re-executions spend much energy. While P is rising, the number of remote executions increases and this saves energy. But since the screen and the wireless module are always turned on during waiting, they cost more energy if the waiting time is prolonged. For this reason, the energy consumption under a given timeout fraction has a minimum. Combining both of the two metrics, the best timeout P is very close to the value we predicted through the model simulation. Therefore, the efficiency of our method to identify the percentage based optimal timeout value is verified.

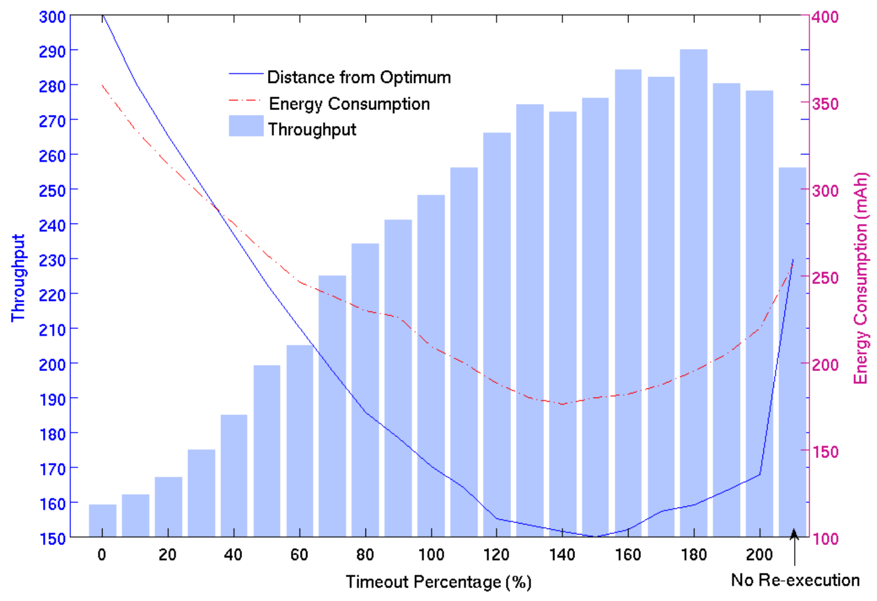


Figure 6.7: Performance of Different Timeout P.

6.4 Summary

In this chapter, we design a Stochastic Activity Network (SAN) model based on the engine structure and three metrics to evaluate the engine performance. The model parameters are set to values obtained from experimental data of our engine. We compare the performance of different local re-execution launching intervals with the model simulation and find the optimum. Then, the optimal launching interval is returned to configure the engine and its efficiency is verified with experiments. The result shows that starting the local re-execution after an appropriate interval provides a better performance than always offloading, and our model can effectively find the optimal launching interval.

Chapter 7

Modular Synthetic Model with Multiple Clients

In this chapter, we design three individual modular static models to respectively emulate the operation of a mobile client, network and remote server in the mobile offloading system. In the practical scenario, the three parts run independently. A series of standard interfaces are used to connect them. But the performance of each part has a significant impact on the whole system. One of the most uncertain factors which affect the system performance is the mobile client. The impact mainly comes from two aspects: the number of clients and their relatively independent behaviours. Normally, the number of clients is not constant. As the traffic in the network and the workload on the server are positive correlated with the number of clients, the change of this number will directly affect the system performance. In addition, the behaviours of individual mobile clients are independent of each other. For instance, the reactions of individual clients are not the same when facing a network failure. The diverse behaviours will also have impact on the system recovery. The motivation of designing the modular models is to analyse the impact of the diverse behaviours. By assembling the individual models together to compose a complete offloading system model, we provide the capability of flexibly adjusting the number of clients in the mobile offloading system.

With these models, we analyse the impact of multiple clients on the failure handling scheme with local restart. We find a congestion avoidance mechanism by repeating local execution when the throughput of a server reaches its upper limit. In this chapter, first the workflow of the congestion avoidance mechanism is introduced. Then, the system model consisting of several modular component models is depicted. Finally, the simulation result demonstrates the efficiency of the congestion avoidance mechanism.

7.1 Congestion Avoidance with Local Execution

As we have introduced in the last chapter, using the local restart can not only save the task completion time but also energy consumption. However in Section 6.1, we assume that the cloud server is perfect as it will always responsed immediately when receiving a request. The request from the mobile client does not have to wait in the queue for service. Unfortunately, this ideal server does not exist in the real computer system. As we know, no matter how fast the processing rate of the server, it cannot simultaneously answer infinite requests. Maintaining a waiting queue is necessary to cache the requests when the server is busy. And obviously, the queue length has an upper limit. Although memory technology has been fast upgraded, it is still difficult to provide the capacity to cache all the request. Because traffic based on Internet Applications also experiences a massive growth at the same time. Within a short interval, when the number of requests arriving exceeds the capacity of queue, the server cannot handle the over load. The later requests have to be discarded since the server has been overloaded. Therefore, with a network failure, a busy server can also cause the failure of the offloading task completion.

When the offloading task has to wait a long time for the service, using local restart can not only speed up the task completion but also reduce the workload of the server. As the mobile client also has computation power, the mobile offloading system is not a pure client-server system. It can be seen as a distributed computation system with a main compute center: the cloud server, and a large number of small compute units, the mobile clients. When the workload of the compute center increases to an unaffordable level, the small compute units can be employed to share the workload.

In fact, when the queue of the server stays full and new requests still come continuously, server congestion happens. It can be predicted that if no congestion avoidance mechanism is applied, the congestion at the server usually lasts for a while. The queue length of the sever cannot be reduced until the mean interval between new arriving requests is longer than the mean service time for a request.

In this thesis, we assume that the sever will not actively notify the clients of its overload. The mobile client can only infer congestion from the long waiting time for task completion. An efficient method to avoid the congestion is reducing the number of new coming requests. In the mobile offloading system, if most of the clients keep executing the offloading task locally instead of delivering them to the server, the number of request arriving at the server will quickly decline. At the same time, executing these tasks locally can also maintain the total throughput of the whole offloading system by avoiding useless waiting time at the server. After the queue length of the server returns to a moderate level, offloading tasks can be pushed to the server again.

Fig.7.1 shows the three states of the congestion avoidance mechanism with local execution.

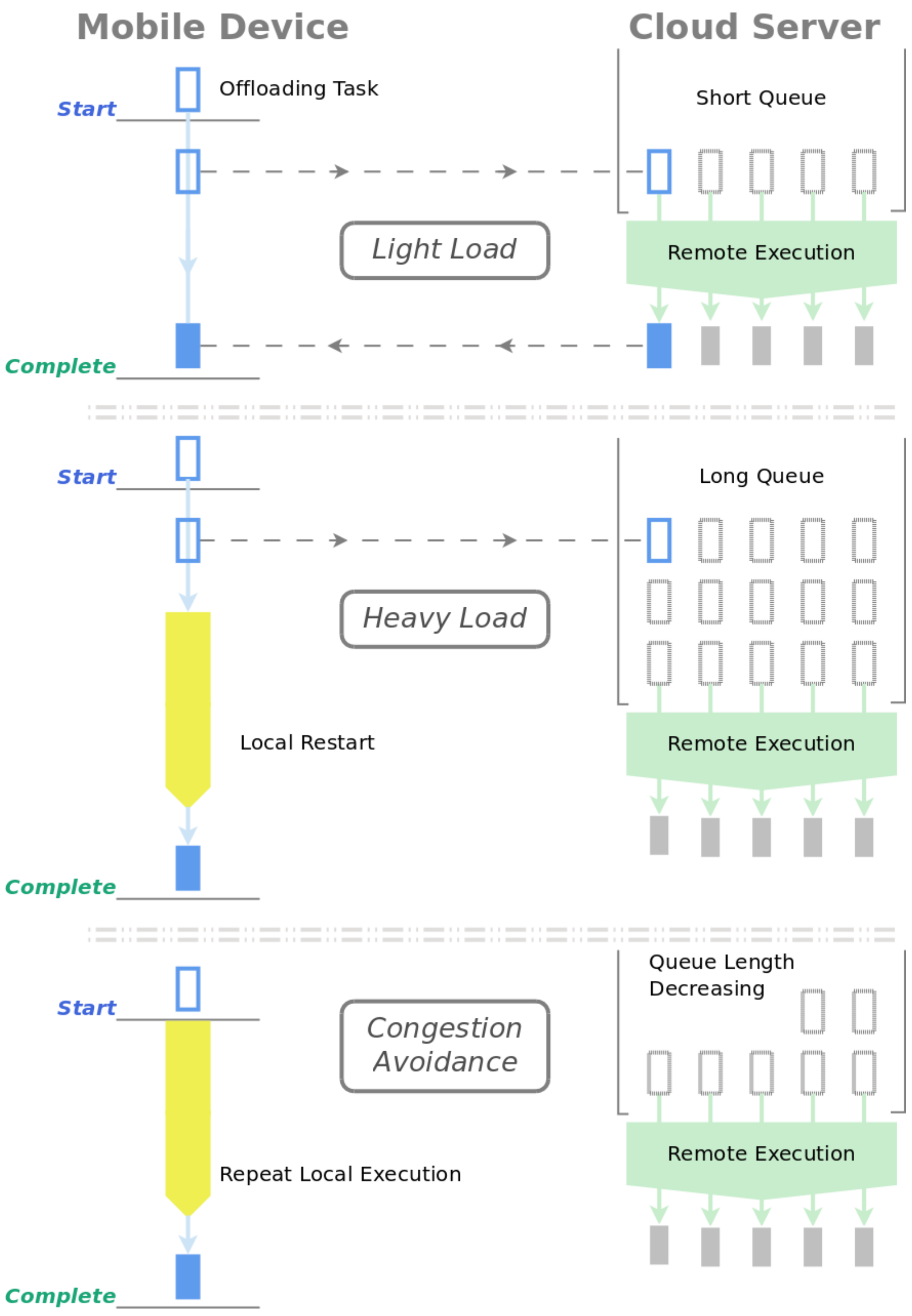


Figure 7.1: Congestion Avoidance by Repeated Local Execution

In the light load state, only a few offloading requests arrive at the server. The cloud server can rapidly complete these tasks and return the results. In the heavy load state, a larger number of tasks wait in a long queue to be completed by the server. As we introduced in section 5.3, when the waiting time exceeds the timeout threshold, the local restart is launched to complete the offloading task. Then the mobile client informs the server to discard the request from the queue. After the first local restart the mobile client has detected the congestion, it continuously uses local computation resource to complete the next several offloading tasks. As shown in the state of congestion avoidance, the mobile device repeats the local execution instead of delivering new offloading requests to the server. Since there are less new coming requests, the server can quickly recover to a normal state by completing the tasks which are waiting in the queue.

7.2 Modular Analysis Models

We still use the Stochastic Activity Networks (SAN) as the modelling technique. According to the individual operation process of client decision, data transmission and server response in the mobile offloading system, three types of model are designed. Among them, the models of client decision and server response are still split into several sub-models according to the different functions of each step in the operation process. First, we introduce the Hyper Erlang Model which represents the delay action in data transmission and task completion. Then the model which emulates the operation of the mobile client is illustrated. At last, we present the complex server model which can be accessed by multiple clients.

7.2.1 Hyper Erlang Model

In completing an offloading task, the process includes several actions which consume time. For example, data transmission, task completion and local execution, all of them need tens of milliseconds to several seconds. Most importantly, according to the experimental experience, the time values are definitely not constant, they are random variables following some given distributions. As we have introduced in Section 3.2.2, the hyper-Erlang distribution is a suitable option to fit these random variables. Fig.7.2 shows the SAN model which can generate a delay action according to a given hyper-Erlang distribution.

Referring to Table A.1, T^{o}_{min} and α are controlled by the activity of *Min_Delay*. The other two parameters m and λ are configured individually into the four activities *Erlang_1—4*. The halting time in the four activities follows an Erlang distribution with two particular parameters: 'shape' m and 'rate' λ. Theoretically, as introduced in [51, 37], this model can be extended to include infinite branches of Erlang distribution, as long as increasing the number of output places of the activity *Min_Delay*. But increasing the Erlang branches, the model spends more time in simulation. The accuracy improvement by fitting a Hyper-Erlang distribution with more

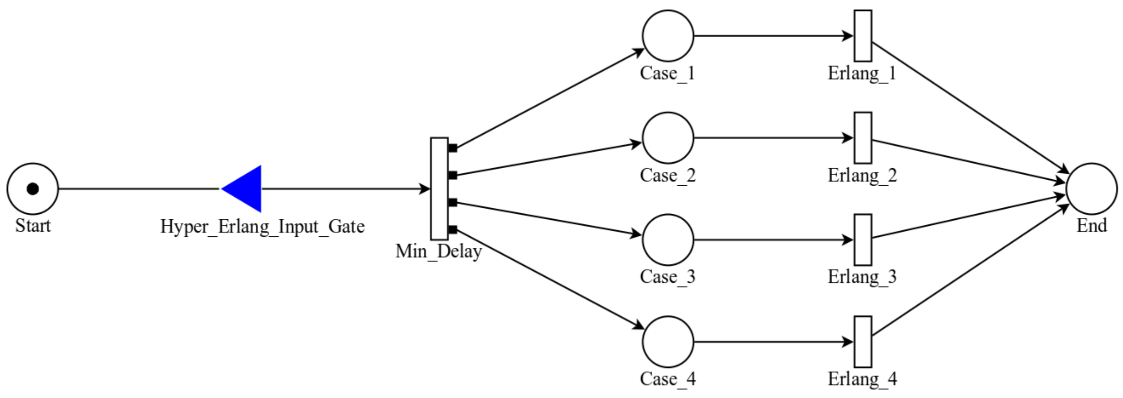


Figure 7.2: Hyper Erlang Model

branches is limited. Thus, we use four branches in this model *Erlang_1—4*, which is accurate enough to represent the distribution of a random variable for the time delay action, as proved by the error measurement in Table A.1 and Table 3.3. The gate of *Hyper_Erlang_Input_Gate* controls the begin of the delay action. When the activity of *Min_Delay* is enabled, only one of the four output places is selected randomly as the destination of the token. And before the token is forwarded by the activity of *Erlang_1—4* to the end place, the activity of *Min_Delay* cannot be enable again.

Table 7.1: Enabling predicate of the input gate *Hyper_Erlang_Input_Gate*

Gate	Predicate	Function
Hyper_Erlang_Input_Gate	$\#Start > 0$ &	$\#Start - 1$;
	$\#Case\_1 = 0$ &	
	$\#Case\_2 = 0$ &	
	$\#Case\_3 = 0$ &	
	$\#Case\_4 = 0$	

# refers to the number of tokens in the given place

7.2.2 Mobile Client Model

As can be seen in Fig.7.3, the model emulates the operation of the mobile client. The two input gates, *Offloading_Input_Gate* and *Local_Execution_Input_Gate*, coordinate together to decide whether the offloading task is delivered to the cloud server or executed locally by the mobile clients. Table 7.2 shows the enable predicates of the two gates. As we have introduced in Subsection 7.1, once the congestion in the server is detected, the number of tokens in the place *Number of Repeat* controls the number of tasks which will be locally completed.

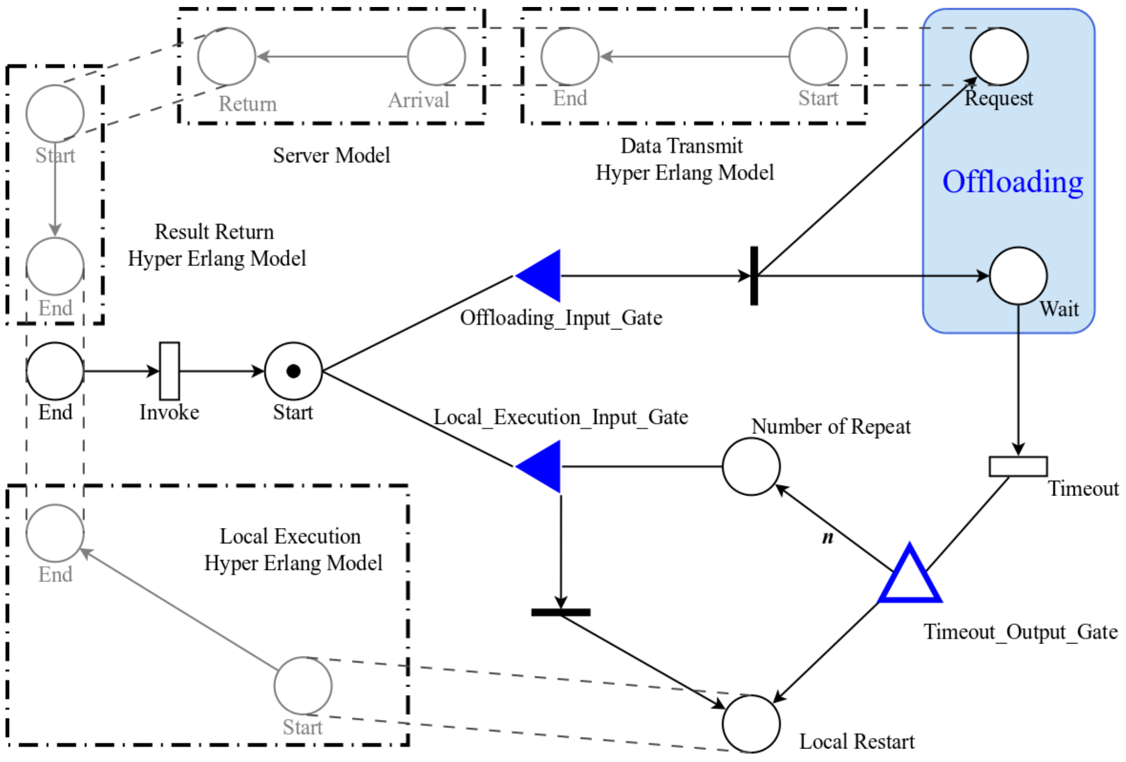


Figure 7.3: Mobile Client Model with Restart

Table 7.2: Enabling predicate of *Offloading_Input_Gate* and *Local_Execution_Input_Gate*

Gate	Predicate	Function
Offloading_Input_Gate	$\#Start > 0$ & $\#Number\ of\ Repeat = 0$	$\#Start - 1$;
Local_Execution_Input_Gate	$\#Start > 0$ & $\#Number\ of\ Repeat > 0$	$\#Start - 1$; $\#Number\ of\ Repeat - 1$;

# refers to the number of tokens in the given place

If the enabling condition of the gate *Offloading_Input_Gate* is satisfied, the mobile client model moves into the offloading state. In this state, both of the places *Wait* and *Request* contain a token. The place *Request* triggers the execution of the following models to complete the offloading task. At the same time, the place *Wait* monitors the waiting time for task completion. Once the waiting time exceeds the timeout threshold, the activity *Timeout* is enabled. The configuration of the activity *Timeout* is the same as its analogue in Fig.6.2 and Table 6.2, so we do not repeat it here.

The output gate *Timeout_Output_Gate* is responsible for launching the local restart and the following congestion avoidance. Table 7.3 demonstrates the output function of *Timeout_Output_Gate*. The parameter n decides the number of offloading tasks which will be locally completed to avoid the congestion in the server. We will analyse the impact of n in the simulation.

Table 7.3: Output Function of *Timeout_Output_Gate*

Gate	Function
Timeout_Output_Gate	#*Local Restart* $=1$;
	#*Number of Repetitions* $=$ n

# refers to the number of tokens in the given place

If a token moves into the place *End*, it means the offloading task has been completed by the server and the result has been received by the mobile device. Then the token in the place *Wait* is removed. After the activity *Invoke* is enabled, a new process of completing another offloading task takes place. The interval between the beginning of the new task and the completion of the old task is based on an exponential distribution that is similar as the analogue in Fig.6.2 and Table 6.2.

7.2.3 Server Model

As shown in Fig.7.4, the server model consists of three sub-models: Arrive Model, Leave Model and Queue Model. Each model respectively emulates a specific function of the cloud server. First, when a new offloading task **J** arrives at the server, it waits in the queue. Then after the server completes all the tasks which are ahead of **J**, **J** is executed. Remote Execution Hyper Erlang Model is used to emulate the execution action. Finally, the completed task leaves the server and returns to the mobile client. This process is emulated by Leave Model. An independent pair of Arrive Model and Leave Model exclusively belongs to every individual mobile client. That means the number of clients equals the number of the pairs of Arrive Model and Leave Model. All the pairs are connected by the unique Queue Model, which is the core of the server model. Queue Model, as the meaning expressed by its name, emulates the process of queueing in the server. The details about each sub-model are introduced below:

Arrive Model

This model uses two instant activities to represent the actions of moving in the queue and waiting in the queue. The two input gates *Start_Input_Gate* and *Execute_Input_Gate* control the two activities. As the enable predicate in Table 7.4 shows, *Start_Input_Gate* controls that each time only one task **J** per client can move in the queue. The place *Previous Queue* records the information about the number of tasks which stand before **J** in the queue. As the output function of *Start_Output_Gate* in Table 7.5 shows, after the instant activity is enabled, the number of

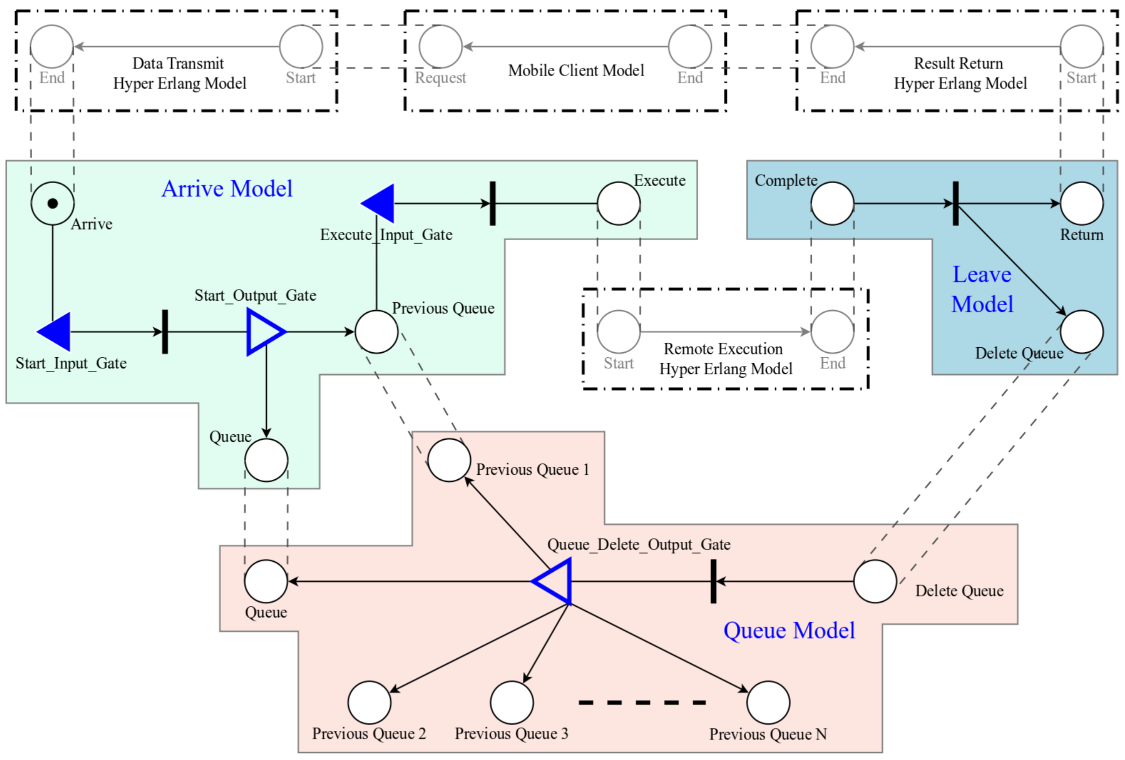


Figure 7.4: Server Model

tokens moved in *Previous Queue* equals the number of tasks which have been in the queue. Then, Arrive Model goes into the waiting state. Once the server completes an offloading task from another client, the number of tokens in *Previous Queue* decreases by one. Until there is only one token left in *Previous Queue*, which means all the tasks standing in front of **J** have been completed, it is the turn of **J** to be executed.

Table 7.4: Enabling predicate of *Start_Input_Gate* and *Execute_Input_Gate*

Gate	Predicate	Function
Start_Input_Gate	$\#Arrive > 0$ & $\#Previous\ Queue = 0$	$\#Arrive - 1$;
Execute_Input_Gate	$\#Previous\ Queue = 1$	$\#Previous\ Queue = 0$;

# refers to the number of tokens in the given place

Leave Model

When the offloading task **J** is completed, Leave Model functions as a hinge to trigger the execution of another two models: Result Return Hyper Erlang Model and Queue Model. The

Table 7.5: Output Function of *Start_Output_Gate*

Gate	Function
Start_Output_Gate	#*Queue* $+ 1$;
	#*Previous Queue* $=$ #*Queue*

# refers to the number of tokens in the given place

former model returns the result back to the mobile client, and Queue Model reduces the queue length of the server.

Table 7.6: Output Function of *Queue_Delete_Output_Gate*

Gate	Function
Queue_Delete_Output_Gate	if(#*Queue* > 0)
	{ #*Queue* $- 1$ }
	if(#*Previous Queue 1* > 1)
	{ #*Previous Queue 1* $- 1$ }
	if(#*Previous Queue 2* > 1)
	{ #*Previous Queue 2* $- 1$ }
	⋮
	if(#*Previous Queue N* > 1)
	{ #*Previous Queue N* $- 1$ }

# refers to the number of tokens in the given place

Queue Model

Queue model stores all the information about the queue length in the server and the position of each offloading task $\mathbf{J}_i$ from client i in the queue. Each time when one offloading task is completed by the server, the instant activity in Queue Model is enabled. The output function of the gate *Queue_Delete_Output_Gate* is listed in Table 7.6. Queue Model uses the places *Previous Queue i, (i = 1,2,...N)* to control the queuing process of each arrived task $\mathbf{J}_i$. The number of tokens in *Previous Queue i* represents the sequence of $\mathbf{J}_i$ in the queue. According to the output function of *Start_Output_Gate* in Arrive Model, *Previous Queue i, (i = 1,2,...N)* represents how many tasks from other clients are before $\mathbf{J}_i$. On the contrary, the unique place *Queue* is shared by all Arrive Models. The number of tokens in *Queue* tells the new incoming offloading tasks the current queue length of the server. Theoretically, we can access an infinite number of Arrive Models to Queue Model. The number of Arrive Models indicates the number of mobile clients in the mobile offloading system. In this chapter, we connect 100 mobile clients to the server. We understand that compared with the real mobile offloading system, 100 clients are still relatively a few. Since more clients consume more time in simulation, we assume that 100 clients are sufficient to analyse the system performance.

7.3 Performance Analysis

We take three steps to explore the efficiency of our congestion avoidance scheme, which utilizes local resource to complete offloading tasks when the server is under heavy workload. First, we design the experiments to collect data for configuring some parameters in our model. Then, the data from experiments is fitted with the Phase-Type distribution by Hyperstar [116]. The fitting results are used to configure the values of the model parameters in simulation. Finally, the simulation results are analysed to demonstrate the efficiency of our congestion avoidance scheme. The optimal configuration of the scheme is also identified.

7.3.1 Measurement Experiment

The offloading task completion consists of three time delay actions: data delivery, remote execution and return of result. In Chapter 3, we take the task completion time as a whole and use it as the metric to evaluate the network quality. In order to measure the time consumed in each of the three actions, we modify the test bed used in Chapter 3. The same mobile device (Samsung GT-S7568) is still used as the client. We move the server side to Google APP Engine [2], which is a real cloud server platform which works as Platform as a Service (Paas). In most situations, the content of offloading tasks are not the same. The time consumption of completing each task is also different. For better simulating the task diversity, we implement a language translation instead of OCR as the sample application into our offloading engine. We translate the famous fiction "Gone with the Wind" from English to Chinese sentence by sentence. The translation of one sentence is one offloading task. On the server side, an online language translation application interface Youdao [4] is utilized to complete the offloading task. On the mobile client side, we install an offline English-Chinese dictionary to translate each sentence. As we mainly focus on the time property of the mobile offloading system, the accuracy of the translation result is not considered in this thesis.

We modify the program of our offloading engine on the server side to record two timestamps: *Arrive* and *Leave*. As shown in Fig. 7.5, *Arrive* is recorded at the arrival instant of the offloading task at the server, and *Leave* is recorded at the moment when the completed result is returned back to the client. The interval between the two timestamps is the time spent in remote execution T_{remote}. This interval T_{remote} is modelled by Remote Execution Hyper Erlang Model as shown in Fig.7.4. For measuring the time used for data transmission, we also record two timestamps on the client side: *Request* and *End*. *Request* is recorded at the moment when the offloading task is pushed out to the server, and *End* is recorded at the moment when the result of the completed

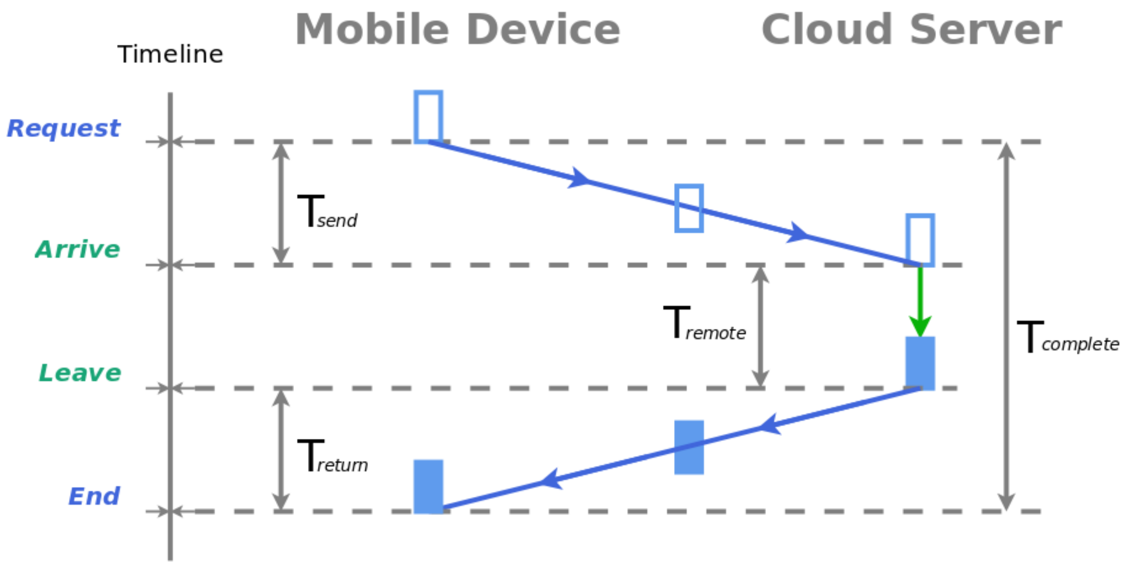


Figure 7.5: Consists of the Task Completion Time

task comes back to the client. The interval between *Request* and *End* is the total offloading task completion time $T_{complete}$. Subtracting T_{remote} from $T_{complete}$, the remainder is the time used in data transmission T_{send} and returning the result T_{return}. T_{send} and T_{return} are modelled by Data Transmit Hyper Erlang Model and Result Return Hyper Erlang Model respectively as appeared in both Fig.7.3 and Fig.7.4. We understand that T_{send} and T_{return} are different. But as our models are mainly interested in the role of the cloud server here, we simply assume that T_{send} and T_{return} are equal as $T_{send} = T_{return} = (T_{complete} - T_{remote})/2$. That means the parameters of the models: Data Transmit Hyper Erlang Model and Result Return Hyper Erlang Model are identical. T_{local} represents the time consumed by the local execution of the offloading task. As shown in Fig.7.3, Local Execution Hyper Erlang Model is used to control T_{local}.

There are total 51,566 sentences in "Gone with the Wind". Thus, we launche 51566 offloading tasks to collect the data. All the tasks are executed consecutively without any interruption. We utilize a 10M Wifi connection (provided by Eduroam [1] in our campus) to link the mobile device and the server of Google APP Engine. In order to guarantee a stable network quality during the experiment, we completed all the offloading tasks on Saturday. As there are only a few students in the campus in the weekend, the network utilization is low and the network quality is good.

7.3.2 Parameter Configuration

We collect data of 51,566 offloading tasks to fit the distribution of T_{remote}, T_{send}/T_{return} and T_{local}. Table 7.7 lists the fitting results and the corresponding parameters in the models. The unit of all time values is milliseconds in Table 7.7. We review the probability density function (*pdf*) of Hyper-Erlang distribution here to help the reader understand the definition of each parameter in our model. For details about Hyper-Erlang distribution, please refer to Section 3.2.2. The *pdf* of Hyper-Erlang distribution $p(x)$ is:

$$p(x) = \sum_{i=1}^{M} \alpha_i \times Er_i(x) \tag{7.1}$$

As we have introduced in section 7.2.1, in our models $M = 4$ and $\sum_{i=1}^{M} \alpha_i = 1$. Er_i in equation 7.1 is the *pdf* of Erlang distribution:

$$Er_i(x) = \frac{\lambda_i^{m_i} x^{m_i - 1}}{(m_i - 1)!} \times e^{-\lambda_i x} \tag{7.2}$$

Generally, m_i is called the shape parameter and λ_i is the rate parameter.

Table 7.7: Results of Distribution Fitting

	Local Execution Hyper Erlang Model: T_{local}		Data Transmit Hyper Erlang Model: T_{send} †		Remote Execution Hyper Erlang Model: T_{remote}	
Activity	Parameter	Value	Parameter	Value	Parameter	Value
*Min_Delay**	T_{min_local}	2.11	T_{min_trans}	0.175	T_{min_remote}	0.583
	α_{local_1}	0.1995	α_{trans_1}	0.3785	α_{remote_1}	0.0482
	α_{local_2}	0.283	α_{trans_2}	0.359	α_{remote_2}	0.4217
	α_{local_3}	0.234	α_{trans_3}	0.109	α_{remote_3}	0.5238
	α_{local_4}	0.2835	α_{trans_4}	0.1535	α_{remote_4}	0.0063
Erlang_1	m_{local_1}	2	m_{trans_1}	235	m_{remote_1}	1
	λ_{local_1}	14.606	λ_{trans_1}	2272.36	λ_{remote_1}	8.0353
Erlang_2	m_{local_2}	119	m_{trans_2}	4	m_{remote_1}	4
	λ_{local_2}	1216.79	λ_{trans_2}	27.679	λ_{remote_2}	199.636
Erlang_3	m_{local_3}	14	m_{trans_3}	13	m_{remote_3}	17
	λ_{local_3}	169.387	λ_{trans_3}	18.045	λ_{remote_3}	1150.98
Erlang_4	m_{local_4}	98	m_{trans_4}	1	m_{remote_4}	15
	λ_{local_4}	866.454	λ_{trans_4}	1.191	λ_{remote_4}	11.3527

* The halting time in the activity *Min_Delay* follows a deterministic distribution.
† The values of the parameters in Result Return Hyper Erlang Model: T_{return} are identical with that in Data Transmit Hyper Erlang Model: T_{send}

There are still two activities *Invoke* and *Timeout* left in the Mobile Client Model that need to be configured. The definition and values of the parameters in the two activity are show in Table 7.8. We control the system workload through $T_{arrival}$ in the simulation. The performance of

Table 7.8: Parameters for the activity *Invoke* and *Timeout*

Name	Distribution	Parameter	Definition	Value
Invoke:	Exponential	$T_{arrival}$	The mean time before the next offloading request arrival.	$0.1s \sim 10000s$
Timeout:	Deterministic	$\underset{\dagger}{1.8} \times (T_{remote} + \underset{*}{2} \times T_{send})$	T_{remote} is the mean of the Hyper Erlang Distribution in Remote Execution Model	0.6s
			T_{send} is the mean of the Hyper Erlang Distribution in Data Transmit Model	0.3s

* Refer to Table 7.7, as the values of the parameters in two data transmission models T_{send} and T_{return} are identical, the numerator is 2.
† According to the conclusion in Section 6.3 and Fig.6.7, the optimal timeout percentage is 180%.

the congestion avoidance scheme under different values of n in the Mobile Client Model are compared through simulation.

7.3.3 Simulation Result

We use Mobius [41] as the tool to execute the model simulation. The simulated system lifetime of a simulation is fixed at 3600s (1 hour), which is long enough compared with the short completion time of a single offloading task. Performance of the congestion avoidance scheme under different workloads are investigated by changing the task arrival interval $T_{arrival}$ in Mobile Client Model. We applied the waiting time at the queue in the server and the throughput of the mobile client as the metric to evaluate the system performance. The queue waiting time is defined as the period which starts at the arrived moment of an offloading task at the server and ends when the task is handled by the server. The congestion intensity of the server can be observed as the queue length, which has a direct impact on the waiting time. Thus, measuring the queue waiting time can provide a clear visualized comparison of the system performance under different configurations of the congestion avoidance scheme.

As shown in Fig. 7.6, the average waiting time of each offloading task in the server queue appears a shape of step increase. The traffic load in the server side increases exponentially with the task arrival interval decreasing in the client side. As the number of clients in our model is still fixed, the queue waiting time has an upper limit when the queue length reach its maximum. Otherwise, the queue waiting time will keep on rising exponentially. Obviously, utilizing the local resource to complete parts of the offloading tasks can reduce the queue waiting time. Congestion occurs when the task arrival rate in the server is larger than its service rate, and the congestion intensity depends on the difference between the two rates. After the offloading task meets a long delay and the mobile client launches local restart to complete this task, using

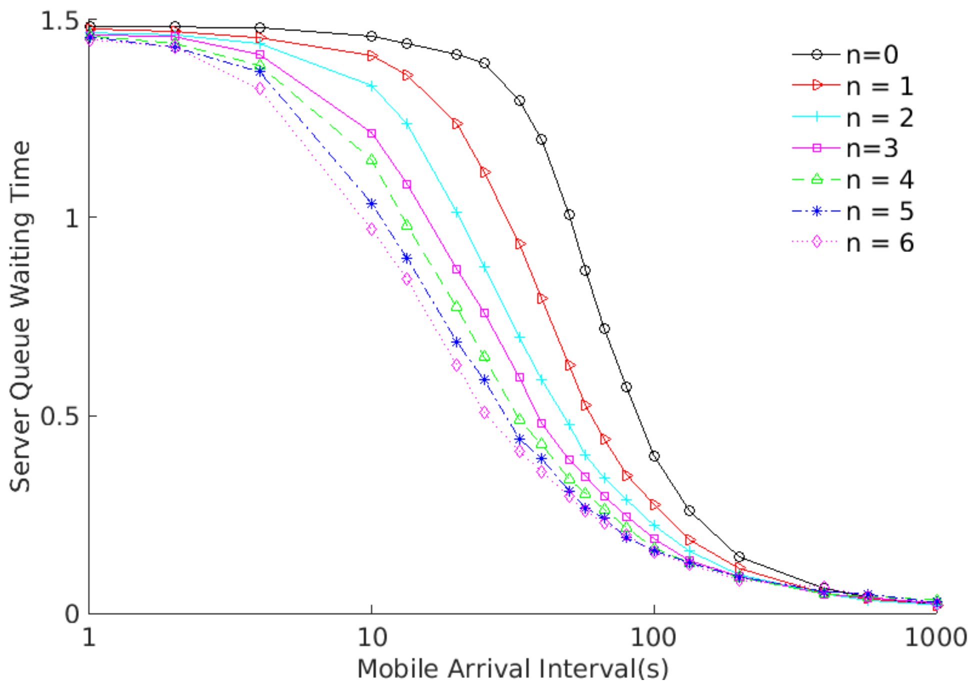


Figure 7.6: Queue Waiting Time under different task arrival intervals

local execution in mobile client to complete the following tasks can narrow the gap. As parts of workload of the offloading tasks are shared by the mobile device itself, the task arrival rate can be kept at a low level.

The efficiency of the congestion avoidance scheme enhances with n, which is the number of local executed tasks. However, the degree of performance improvement declines with the increment of n. From Fig.7.6, we can find that when $n > 3$, the queue waiting time cannot be reduced much further. One reason could be that the gap between the task arrival rate and the service rate widens too fast for the congestion avoidance to offset. Another reason may arise from the categorised mobile clients after they experience the offloading failure. Assume that when congestion occurs, some of the clients will experience offloading failures and launch local restart to complete the task. We define these clients as type **A**. The other clients whose tasks are smoothly completed by the server are defined as type **B**. When n increases, type **A** clients need a long time to finish several local executions and launch offloading again. In this period, the traffic of the server is low and the queue length is acceptable. Thus, type **B** clients can keep using the server to complete the offloading task in a short time. When the next time **A** clients launch offloading again, they still face a higher possibility of a long waiting time in the server queue than type **B** clients. The categorised clients may impair the efficiency of the congestion avoidance scheme.

Fig.7.7 shows the throughput of the mobile client in the system lifetime of simulation. As

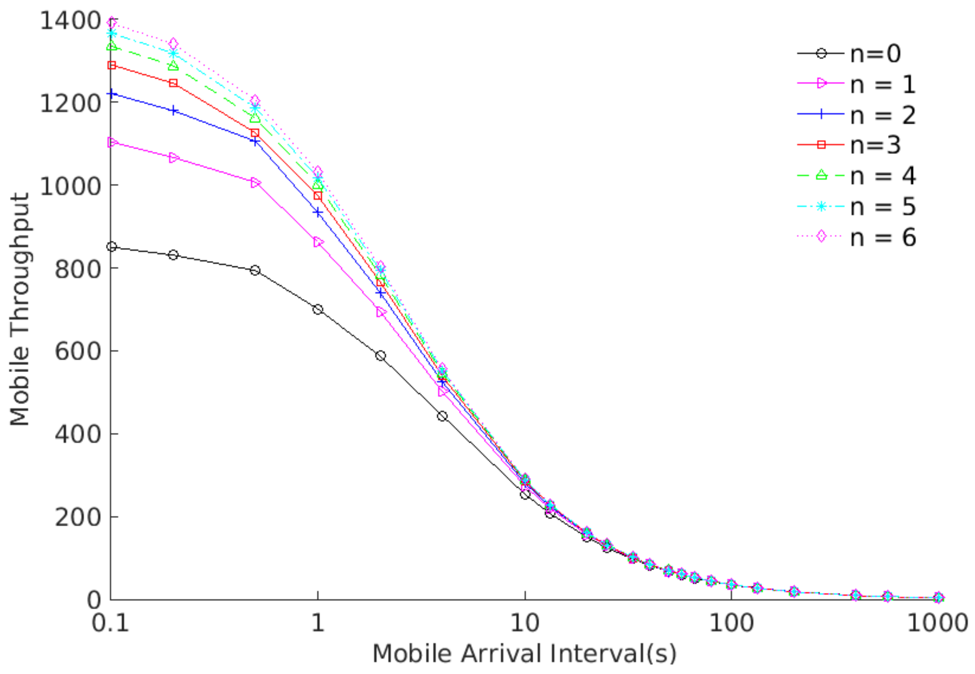


Figure 7.7: Throughput of Mobile client with different task arrival intervals

the throughput increases with n under the same task arrival rate, the efficiency of the congestion avoidance scheme is shown. Using congestion avoidance can at least enhance the throughput by around 40%, as the throughput of $n = 0$ is 836 but the throughput of $n = 1$ is 1108. As similar as Fig.7.7, the throughput improvement also has an upper bond as the throughputs of $n > 2$ are close. It demonstrates that utilizing too many times of local execution can trade for only a little performance improvement. Although locally executing the offloading task can alleviate the intense traffic in the server, the throughput of the mobile client is restricted by its low computation capability. As we have not considered the power consumption of mobile clients in using the local execution, the cost of the congestion avoidance scheme with $n > 2$ could be too expensive to be applied. Therefore according to the simulation result, we propose an optimal congestion avoidance scheme with launching two local executions after the offloading task experiences a failure.

7.4 Summary

In this chapter, we proposed a congestion avoidance scheme for the mobile offloading system. By employing local execution to share the high workload of the server, the whole throughput of the system is increased. For evaluating the performance of the congestion avoidance scheme, we designed a system model with SAN model. The system model consists of three types of modular models, which respectively emulate the operation of a mobile client, data transmission and cloud server. We use the experimental data to configure the parameters in our model for simulation. The simulation results demonstrate that our congestion avoidance scheme can effectively reduce the queue waiting time in the server.

Chapter 8

Optimal Dynamic Restart Mechanism

The restart algorithm mentioned in Chapter 4 relies on the probability density function *pdf* of the task completion time. In pratice, a density function is approximated by the corresponding histogram. Since the distribution of the completion time in a real-time system keeps changing with the operation, a dynamic method to adapt the histogram is required. In [45] the bucket width is adjusted when the number of samples in some buckets satisfy a given criterion. Histogram data can be stored in a structure called Q-digest which is a binary tree [128]. This allows to quickly find quantiles of the data set using a post-order traversal on the tree. In [58] the data stream is compressed by wavelet transform into a sketch. The quantile query is answered by estimating the original data with the sketch. As introduced in [131, 96, 72, 145, 144], wavelet transform is an efficient method used for histogram compression. But as wavelet transform is a heavy computation task to the mobile client considering its restricted resource, we cannot implement wavelet transform in our restart scheme.

There are also a large number of methods that have been proposed to compress the massive data and fast the query process [12, 73, 132, 74, 79, 154]. In [73], an anti-entropy aggregation protocol is proposed to compute aggregates of components properties like extremal values, average and counting. And [132] focused on the conditional expectation of quantile derivative of a linear combination of random variables. The authors of [74] revealed the relevant dependability issues of the majority of the existing aggregation algorithms which are used to allow the determination of meaningful properties. Based on iterative averaging techniques, the authors proposed two new Push-Pull Gossip algorithms to solve the problem. In [79], an efficient partitioning algorithm is introduced to break an array into some intervals and keep the maximum weight of the intervals is minimized.

All these methods can be used to set up the histogram for the restart algorithm. We do not evaluate the different algorithms in this chapter. We use a width-fixed histogram and propose a cost-effective method to update the histogram at run time.

8.1 Dynamic restart scheme

The procedure of fitting a theoretical distribution and computing the optimal restart timeout from this distribution is very expensive in terms of computation cost. Various algorithms and tools exist for fitting PH distributions to empirical data [133, 64, 136, 149, 16], and the fitted distributions approximate the data in many cases very well. For efficiency reasons we use a direct method [117] to estimate $g(\delta)$ and $E[T]$ from the histogram. We dynamically build and update a histogram and then repeatedly determine the optimal restart timeout as discussed in the following subsections.

The procedure of fitting a theoretical distribution and computing the optimal restart timeout from this distribution is very expensive in terms of computation cost. For efficiency reasons we use a direct method [117] to estimate $g_n(\tau)$ and $E[T_i]$ from the histogram. The asymptotically unbiased ratio estimator is introduced in Subsection 8.1.2. We dynamically build and update a histogram and then repeatedly determine the optimal restart timeout.

8.1.1 Dynamic Histogram

A histogram simply divides up the range of possible observations into intervals, which we call buckets, and counts the number of observations that fall into each bucket. Buckets can have a variable or a constant width; we choose the latter for simplicity. Histograms initially hold too few samples to provide a good approximation of a probability distribution. After collecting data for a while a stationary distribution is represented increasingly well. However, if the distribution changes, old samples will never be dismissed from the histogram and will forever bias the new probability distribution.

There are several options how to handle changes in distribution: the histogram can be repeatedly flushed as to build up a new histogram for the respective current state of the system. This introduces many initial periods with insufficient data. Another option is to transform the buckets into *dripping buckets* that lose samples constantly over time. It is not easy to adjust the dripping speed such that the histogram will hold sufficient but not too many samples at all times [107, 97, 128, 134, 70].

We propose a partial flush which is tuned using two parameters, the total number of samples in the histogram when executing the partial flush and the percentage of samples to equally flush from all buckets.

Algorithm 1 shows the algorithm to initialise the histogram prior to run time. The parameters are the following:

T^o_{min}: The lower bound of the histogram.

T^o_{max}: The upper bound of the histogram.

T_l: The task completion time by local execution.

Algorithm 1 (Initialization for the histogram)

```
$T_l \leftarrow$ Local_Run()   //Complete the task by local execution
$T^o_{min} \leftarrow$ Offload_Run()   //Complete the task by offloading
$T^o_{max} = T_l$
$\triangle_B = (T^o_{max} - T^o_{min})/N$   //$\triangle_B$: The bucket width
for $i = 1$ to $N$ do
    $B_{average}[i] = 0$
    $N_B[i] = 0$
end for
$N_{out} = 0$
$B_{out} = 0$
```

N: The number of buckets in the histogram.

$B_{average}[i]$: The mean of all the samples in the ith bucket.

$N_B[i]$: The number of samples in the ith bucket.

N_{out}: The number of samples, whose value $> T^o_{max}$.

B_{out}: The mean of all the samples $> T^o_{max}$.

The number of buckets N must be chosen manually. The upper bound of the histogram is determined by the execution time of one local run. The lower bound is given as the execution time of one offloading task. In the course of the experiments there may later be shorter offloading times which will be used as new lower bound and additional buckets will be inserted. These choices are motivated by the purpose of the histogram: to determine the optimal restart timeout the precise shape of the distribution in the tail is not needed.

Algorithm 2 shows the algorithm to record a new sample at run time. If the sample comes from local execution, T_l is updated by the mean of its original value and the new sample. Hence, the impact of old samples is reduced and replaced by that of new ones.

If the new sample is produced by offloading, it can be added to the histogram in three ways according to its value. **Case 1**, when new samples are larger than T^o_{max}, they are all added to the *out* bucket. **Case 2**, when a shorter offloading time arrives, M additional buckets are inserted, M is calculated based on the ceiling function shown in line 9. T^o_{min} moves down to include the new sample. Line $21 \sim 24$ adjusts the mean and index of each original bucket accordingly. **Case 3**, when the sample falls into the range between T^o_{min} and T^o_{max}, it is added to the corresponding bucket in the histogram. Fig. 8.1 is the illustrative diagram of the three cases.

The partial flush algorithm, shown as Algorithm 3, needs the two new parameters N_{bound} and p:

N_{bound}: threshold to start the update. When the number of samples stored in the histogram exceeds this value, the update algorithm is triggered.

p: percentage of samples to be kept. From each bucket, $(1-p)/100 * n_i$ samples are removed

Algorithm 2 (Recording a new sample)

```
Local Execution:
 1: $T_{temp} \leftarrow$ Local_Run()
 2: $T_l = (T_l + T_{temp})/2$
Offloading:
 3: $T_{temp} \leftarrow$ Offload_Run()
 4: switch $T_{temp}$ do
 5:     case $1 : T_{temp} \geqslant T^o_{max}$
 6:         $B_{out} = \frac{(B_{out} \times N_{out}) + (T_{temp} - T^o_{min})}{N_{out}+1}$
 7:         $N_{out}++$
 8:     case $2 : T_{temp} < T^o_{min}$
 9:         $M = \lceil (T^o_{min} - T_{temp})/\triangle_B \rceil$
10:         INSERT($M$)
11:         $B_{average}[1] = T_{temp} - T^o_{min}$
12:         $N_B[1] = 1$
13:     case $3 : T^o_{min} \leqslant T_{temp} < T^o_{max}$
14:         $j = \lfloor (T_{temp} - T^o_{min})/\triangle_B \rfloor + 1$
15:         $B_{average}[j] = \frac{(B_{average}[j] \times N_B[j]) + (T_{temp} - T^o_{min})}{N_B[j]+1}$
16:         $N_B[j]++$
17:
18: function INSERT($k$)
        //Insert $k$ empty buckets between $T_{temp}$ and $T^o_{min}$
19:     $N = N + k$
20:     $T^o_{min} = T^o_{min} - \triangle_B \times k$
21:     for $i = 1$ to $N$ do
22:         $B_{average}[i+k] = B_{average}[i] + \triangle_B \times k$
23:         $N_B[i+k] = N_B[i]$
24:     end for
25: end function
```

Algorithm 3 (Update for the histogram)

```
$B = \sum_{i=1}^{N} N_B[i] + N_{out}$
if $B > N_{bound}$ then
    $N_B[i] = \lfloor N_B[i] \times p \rfloor$     // $i$ from 1 to $N$
    $N_{out} = \lfloor N_{out} \times p \rfloor$
end if
```

if the bucket holds a total of n_i samples before the partial flush.

A large number of samples N_{bound} until partial flush leads to a long sampling period. Conversely, a large percentage p indicates that the majority of the samples are kept after updating. This will lead to frequent inexpensive partial flushes. Please note that the mechanism is related to hysteresis as used in the control of queueing systems.

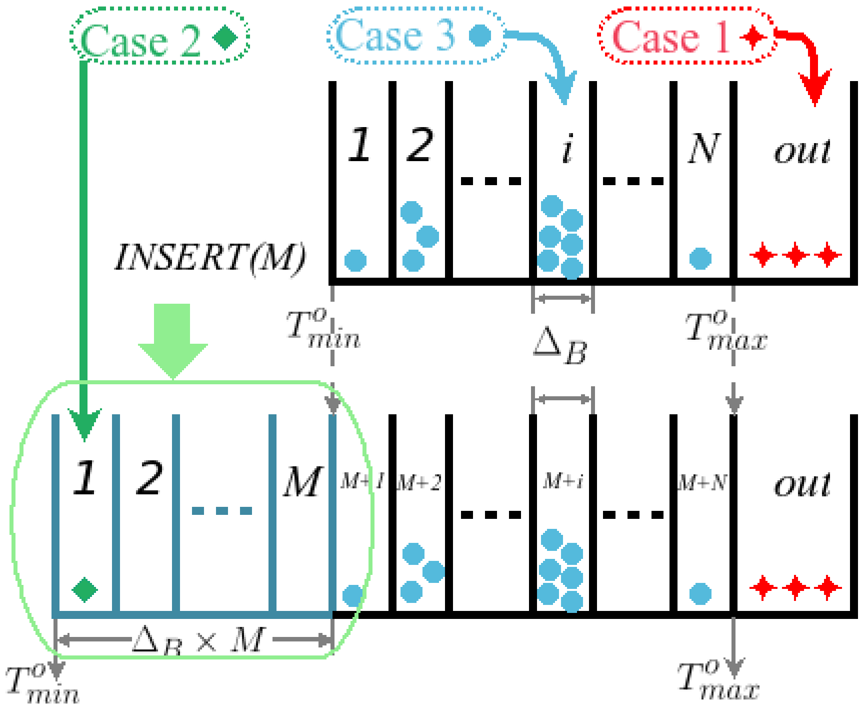


Figure 8.1: Recording a new offloading sample

8.1.2 Asymptotically Unbiased Ratio Estimator

The estimate for the optimal restart timeout is based on the asymptotically unbiased ratio estimator [36]. Using the dynamic histogram proposed in the last subsection, an estimator for $g(\delta)$ in equation (4.9) from Chapter 4 is:

$$\hat{g}(\delta_i) = \frac{\sum_{j=i}^{N} N_B[j] \cdot B_{average}[j] + N_{out} \cdot B_{out}}{(\sum_{k=i}^{N} N_B[k] + N_{out})(1 - \hat{F_o}'(\delta_i))} - \delta_i \tag{8.1}$$

We assume that the optimal timeout δ only takes on values $\delta_i = i \times \triangle_B, i = 1, 2, ..., N$. The cumulative distribution function $\hat{F_o}'(\delta_i)$ is estimated as:

$$\hat{F_o}'(\delta_i) = \frac{\sum_{j=1}^{i} N_B[j]}{\sum_{k=1}^{N} N_B[k] + N_{out}} \tag{8.2}$$

If the maximum estimate $\hat{g}(\delta_i)_{max} > T_l$, the local restart condition (4.8) is fulfilled. Then, an estimate of $E[T]$ provides the optimal timeout.

$$\hat{E}[T]_{\delta_i} = \hat{M}'(\delta_i) + (1 - \hat{F_o}'(\delta_i))(\delta_i + T_l) + T^o_{min} \tag{8.3}$$

Remember that we have shifted all data, and the histogram to the origin. Therefore the lower bound T^o_{min} of the histogram should be added to the expectation. The partial moment $\hat{M}'(\delta_i)$ is estimated as:

$$\hat{M}'(\delta_i) = \frac{\sum_{j=1}^{i} N_B[j] \cdot B_{average}[j]}{\sum_{k=1}^{i} N_B[k]} \tag{8.4}$$

The optimal local restart time can be identified by selecting the value of δ_i, which minimizes $\hat{E}[T]_{\delta_i}$, and the optimal timeout is $\tau = \delta_i + T^o_{min}$. Actually, at run time first the restart condition is evaluated and if it is not satisfied, $\hat{E}[T]_{\delta_i}$ is not determined.

For the hybrid adaptive restart scheme, an estimator for $g_n(\tau)$ in equation (4.27) and $E[T]$ in equation (4.19) are:

$$\begin{aligned}\hat{g}_n(\tau_i) =& \frac{\sum_{j=i}^{N} N_B[j] \cdot B_{average}[j] + N_{out} \cdot B_{out}}{(\sum_{k=i}^{N} N_B[k] + N_{out})(1 - \hat{F_o}'(\tau_i))^n} \\ &- \sum_{l=1}^{n-1} \frac{\hat{M}'(\tau_i) + \tau_i}{(1 - \hat{F_o}'(\tau_i))^l} - \tau_i \\ &- T^o_{min}(1 - \frac{1}{\hat{F_o}'(\tau_i)})(1 - \frac{1}{(1 - \hat{F_o}'(\tau_i))^{n-1}})\end{aligned} \tag{8.5}$$

$$\begin{aligned}\hat{E}[T]_{\tau_i} =& \hat{M}'(\tau_i) + \sum_{k=1}^{n-1} (1 - \hat{F_o}'(\tau))^k (\hat{M}'(\tau_i) + \tau_i) + (1- \\ & \hat{F_o}'(\tau_i))(\tau_i + T_l) + \frac{1 - (1 - \hat{F_o}'(\tau_i))^n}{\hat{F_o}'(\tau)} T^o_{min}\end{aligned} \tag{8.6}$$

The optimal local restart time is identified by selecting the value of τ_i, which minimizes $\hat{E}[T]_{\tau_i}$, and the optimal timeout is $\tau = \tau_i + T^o_{min}$.

8.2 Evaluation of Single Local Restart

In order to evaluate the performance of the dynamic local restart scheme, it is implemented in our mobile offloading engine [146] and evaluated using the OCR application with the same picture as Fig. 3.1. As introduced in Section 3.2, we again conduct measurements over a period of 24 hours from 8:00 on 28th April 2014 and we sampled 54 318 completion times. Using the experiment we then show that our dynamic histogram captures changes in the system and allows the offloading system to react to those in real-time.

Fig. 8.2 shows a short episode of the whole experiment process. This episode lasts for about 5 minutes (begins at 9:12) and contains 180 successive tasks. A scatter plot of some related parameters of the 180 tasks is shown in Fig. 8.2. It can be seen that the potential benefit of the local restart, $\hat{g}(\delta)_{max}$, first increases stepwise and then remains constant. After some very long

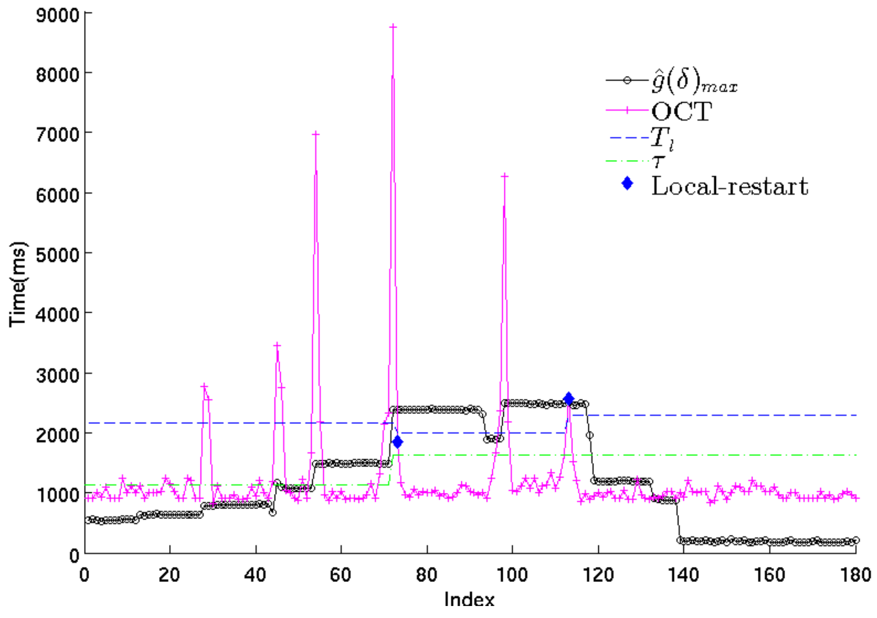


Figure 8.2: Scatter plot of the dynamic local restart scheme with $N = 20$, $N_{bound} = 100$ and $p = 50\%$.

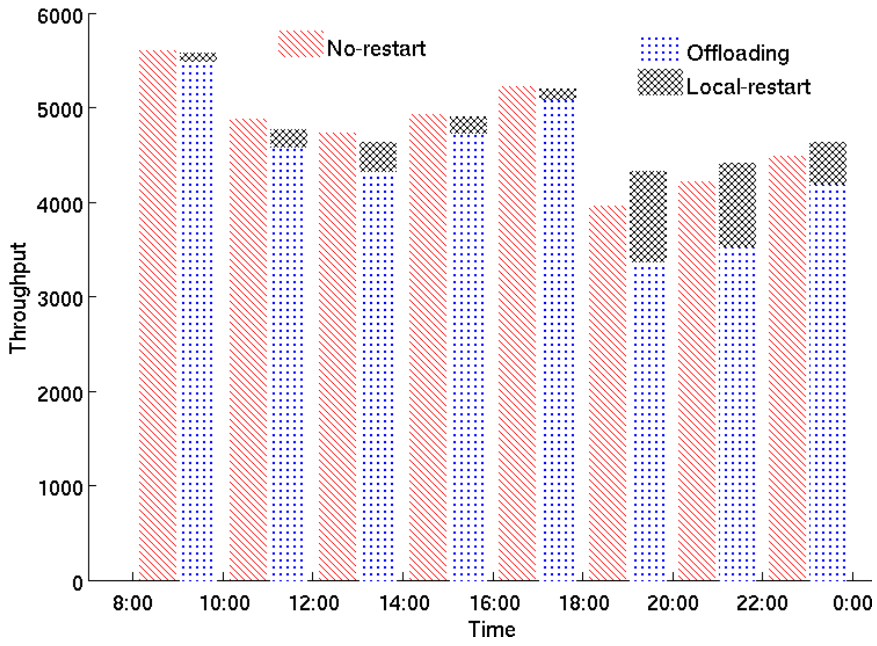
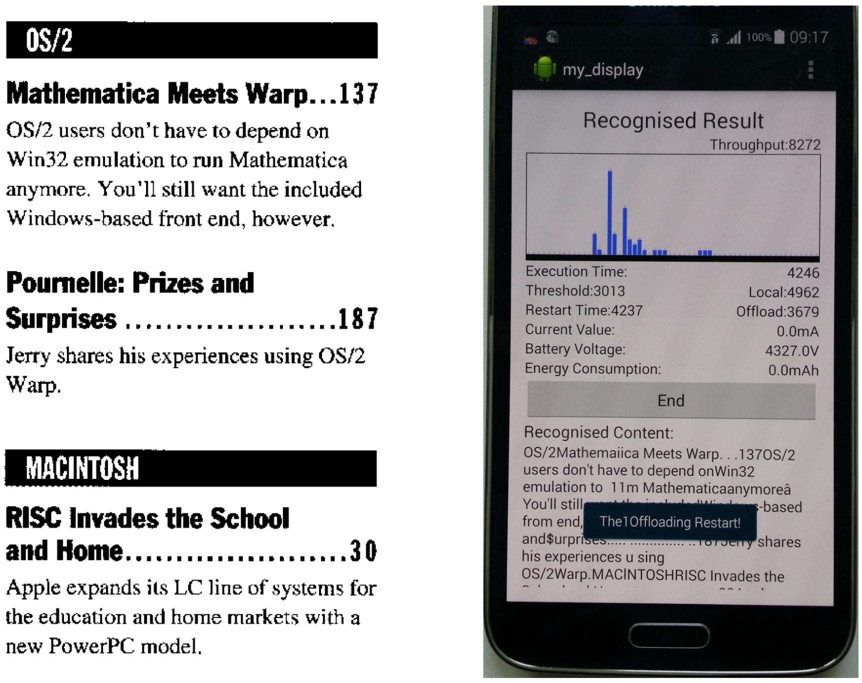


Figure 8.3: Throughput of different times in a day

offloading times, $\hat{g}(\delta)_{max} > T_L$, several restarts complete the computation locally.

For comparing the performance of the scheme with and without the dynamic local restart, the throughput of the two schemes over periods of two hours are shown in Fig. 8.6. We define


OS/2

Mathematica Meets Warp...137

OS/2 users don't have to depend on Win32 emulation to run Mathematica anymore. You'll still want the included Windows-based front end, however.

Pournelle: Prizes and Surprises 187

Jerry shares his experiences using OS/2 Warp.

MACINTOSH

RISC Invades the School and Home......................30

Apple expands its LC line of systems for the education and home markets with a new PowerPC model.

09:17

my_display

Recognised Result

Throughput:8272

Execution Time: 4246
Threshold:3013 Local:4962
Restart Time:4237 Offload:3679
Current Value: 0.0mA
Battery Voltage: 4327.0V
Energy Consumption: 0.0mAh

End

Recognised Content:
OS/2Mathemaiica Meets Warp. . .137OS/2 users don't have to depend onWin32 emulation to 11m Mathematicaanymoreâ You'll still [illegible]-based from end, The1Offloading Restart! and$urpri[illegible] shares his experiences u sing OS/2Warp.MACINTOSHRISC Invades the

Figure 8.4: The image to be recognised and the mobile Application

the throughput as the number of tasks completed in each period. Here we compare data from the two experiment sessions that took place on different days: the right column in each interval represents the first series of experiments without restart, while the left column shows the new series of experiments using restart. Surprisingly, both columns follow a similar pattern over the day and in most intervals the throughput is almost identical in both experiment series. Only for the last three pairs of columns the dynamic local restart scheme can effectively increase the throughput.

In conclusion, the dynamic local restart scheme can effectively increase the system performance sometimes and does not harm it at any time.

8.3 Comparison of Adaptive Restart

In order to observe and analyse the performance of the adaptive restart scheme in a real application we design an experiment. Using the experiment we show a comparison of the three restart modes as introduced in Section 4.3 using the same scenario.

8.3.1 Experiment Configuration

For the experiments a mobile phone (Samsung S5-G900F, Android 4.4.2) and a server (4 cores: Intel Xeon CPU E5649 2.53GHz) have been used. The mobile phone is placed in a main teaching building (Fig. 8.5 shows the floor plan of the building) and connects to the Internet through Wifi (100Mbps provided by Eduroam). The server is in the lab of the university campus and connects to the Internet through a LAN port of 100M. We have used the Linux command ”traceroute” to track the route from the mobile phone to the server. Normally, the route passes 6 hops to reach the destination, and the total round-trip time is around 8ms. The offloading engine as introduced in Chapter 5 includes an Android Application (App) for the mobile client and a website project for the server. In our experiment, the Tesseract OCR Engine [3] is used in both parts of the offloading engine. An image (400×680px, 6 KiB) with a rectangle text region, as shown in Fig. 8.4, is used for image recognition. Only 100 Bytes are used to represent the decyphered words.

In order to imitate a scenario where the network connection is unreliable, we walk around the main building carrying the mobile device. The blue line in Fig. 8.5 shows the route of the device. As shown by the scale on the upper right corner in Fig. 8.5, the area of this teaching building is about $300 \times 200m^2$ and seamlessly covered by wireless network. When the mobile device is moving, it has to hand-off frequently between different Wifi access points(AP). We state that the time interval of switching from one AP to the next is decided by some factors, e.g. the moving velocity of the device, the idle capacity of the next AP and the number of available APs near the device. But in our experiment, we assume that this interval time is a random variable. During the interval, the wireless network is unavailable for the mobile device. Thus the wireless network connection between the mobile device and the server is unstable when the device is moving.

Completion of an offloaded OCR task can be divided into three phases: 1) the Android application transmits the image from the mobile device to the server, 2) the words on the image are recognised using the OCR engine in the server, and 3) the mobile device receives and displays the result from the server. The Offloading Completion Time is the time needed to complete the three steps. The same offloading task has been repeated successively while the mobile device was moving. The results were stored in a text file in the mobile device. The memory of the mobile phone used for caching is cleared after each task completion and reused again in the next new task.

8.3.2 Experimental Results

In our experiment, we started from point *A* as shown in Fig. 8.5. For initialising the dynamic histogram, we stayed there for five minutes. Then, we walked ten minutes to the point *B*. A second chronograph is used to measure the time in the experiment. Although we cannot accurately reached the point *B* on time, we guaranteed that the deviation is no more than 15 seconds. Then we had a rest at the point *B* for five minutes. During the break, the mobile device connected with an identical Wifi AP which covers the area around point *B*. Even when we move in the $5 \times 5m^2$ area around *B*, the mobile device did not hand-off to another AP. Thus, in the remaining time,

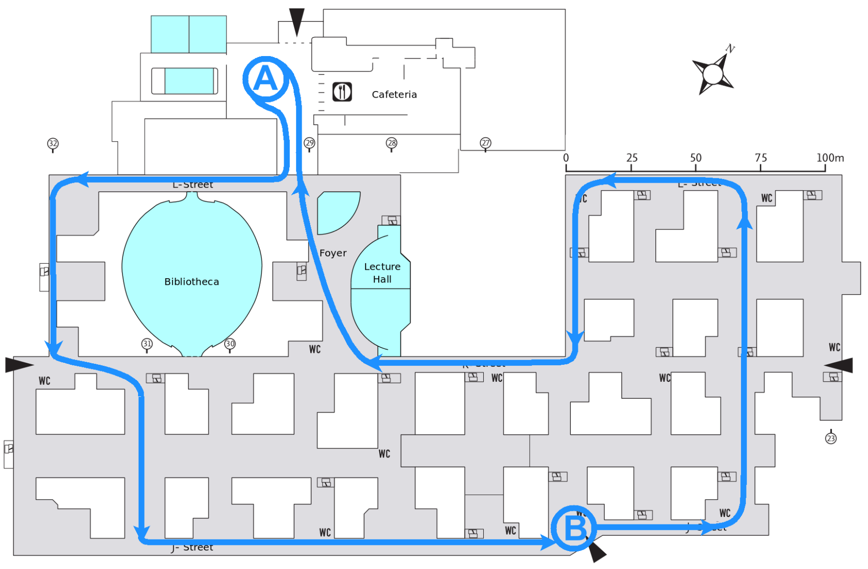


Figure 8.5: The plan of the teaching building

the mobile device kept a reliable network connection to the server. The reason we stayed was to observe the impact of restart on a good network connection in the offloading system.

After the break, we spent ten minutes again walking along another route back to point A as shown by the arrow in Fig. 8.5. The total time of one walk was thirty minutes. We compared five different restart schemes:

A: No restart.

B: Infinite offloading restart, $n \to \infty$.

C: Exclusively local restart, $n = 1$.

D: One offloading restart + local restart, $n = 2$.

E: Two offloading restart + local restart, $n = 3$.

For each scheme we walked along the same route for six times and summed up the results. The throughput of the five schemes over periods of five minutes are shown in Fig. 8.6. We defined the throughput as the number of tasks completed in each period. For each scheme, the number of tasks completed by the original offloading (oof), the restarted offloading and the local execution are marked individually in Fig. 8.6. Surprisingly, the experiment result shows that infinite offloading restart is not the optimum as its throughput is less than the other three restart schemes C, D and E. The explanation for this phenomenon is that when the mobile device is changing Wifi AP, sometimes the hand-off process requires tens of seconds. In particular, if the next access point has already connected to a large number of users, the new coming mobile device is hardly assigned sufficient resources to build a stable connection. Connecting to a heavily loaded AP means that the hand-off time may extend to several minutes. During this time, repeatedly

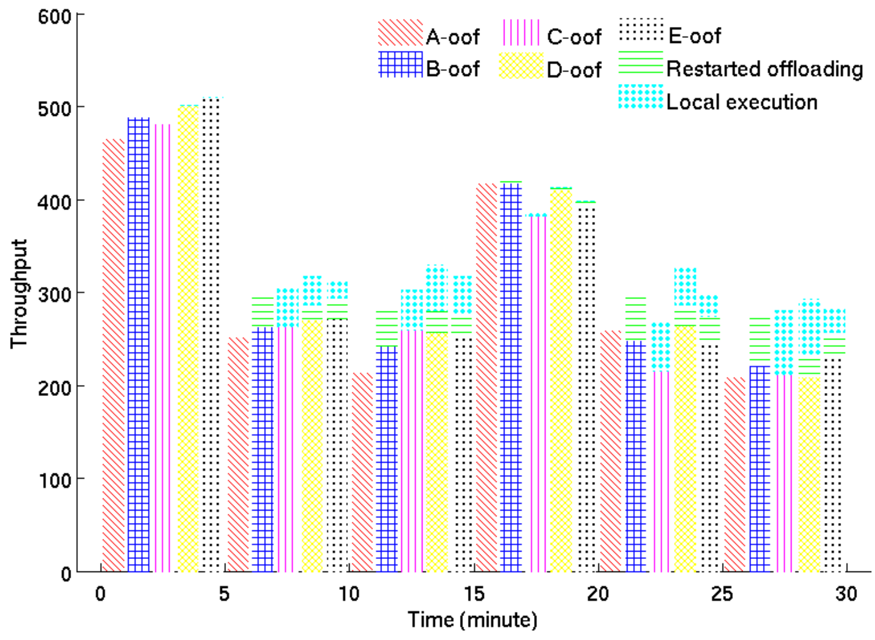


Figure 8.6: Throughput of different times in a day

restarting to offload cannot speed up the task completion. Under a slow hand-off, local restart can at least guarantee task completion.

Fig. 8.6 also demonstrates that the throughput of scheme *C* is lower than that of scheme *D, E*. This phenomenon follows **Theorem 2.** in Section 4.3. However, the throughput of scheme *D* is higher than that of scheme *E* because in practical applications successive offloading tries are not independent. Generally, the failure of the first offloading restart indicates a high possibility of the failure of the successive offloading restart. Here, we demonstrated the correlation between successive restarts through our experiment, but due to its complexity we did not consider the correlation in our theoretical analysis. In conclusion, the automated restart with one offloading retry and one local restart is the optimal scheme to increase system performance.

8.4 Summary

In this chapter, We proposed a dynamic restart scheme for the mobile offloading system. In this scheme, a dynamic histogram is used to track the variation of the network quality, and the restart condition and the optimal time is estimated with the histogram. The efficiency of the restart theory introduced in Chapter 4 is confirmed by experimental results. The experiment shows that adaptively utilizing offloading and local restart at the right time achieves better performance than always offloading.

Chapter 9

Conclusion and Future Work

In brief, offloading is an efficient technique to overcome current constraints of mobile devices. With migrating the heavy computation tasks to remote servers, it can not only provide a better user-experience by accelerating the execution time but also save energy for the mobile device. Connecting mobile terminals and remote servers, the network plays an important role in the offloading system. To some degree, it directly affects the performance of the offloading execution. But smooth offloading of computation depends on a fast and stable network connection, which guarantees seamless communication.

In this thesis, the main target has been to study the efficiency of a failure handling scheme with restart in mobile offloading systems. Since an unstable network may cause a failure or a long delay in the offloading task completion, restart can accelerate the recovery from failure and reduce the task completion time. The stochastic models and iteration based mathematical derivations are used to identify the optimal timeout for launching a restart. The performance of the restart mechanism under different timeouts is evaluated through simulations and experiments. The results indicate that launching restart at a proper moment is applicable to accelerate the task completion and enhance the throughput of the whole system.

9.1 Conclusion

When the offloading task fails, there are two major failure handling schemes. The client may retry offloading or restart the task using the resources in the local device instead of those in the Cloud. The first one is a halt in the current execution state and waiting for the network condition to satisfy the offloading requirement, then relaunching the offloading task. The other one is as soon as the wireless connection is lost, the mobile device immediately re-executes the pre-determined offloading task locally. However, both of the two schemes only adapt extreme scenarios. The former one performs well when the wireless network can quickly recover to the demanding level. But it may waste a large amount of time for waiting if the network suffers from a long time failure. In this case, the latter should be chosen. We have analysed the performance of locally re-executing the offloaded tasks for handling connection failure in mobile offloading, we also proposed the method to find the appropriate moment for launching the local re-execution.

First, we introduced an experiment in Chapter 3 to illustrate the impact of unreliable network on the completion of offloading task. Then, conditions for applying local restart and the optimal timeout in order to reduce the task completion time are mathematically derived based on a greedy method in Chapter 4. Restarting the task again at the appropriate moment can reduce its completion time. We also proposed an adaptive restart scheme to improve the performance of mobile offloading systems. The adaptive scheme restarts first with offloading and when the number of offloading restarts exceeds a given threshold, the task is completed locally in the mobile device. By theoretically comparing the performance of applying different numbers of offloading retries, infinite offloading restart is proved to perform best. However in practice, applying a limited number of offloading restarts is preferred.

In the following Chapter 5, we further introduced the program engine, which is developed to implement the offloading application between mobile devices and cloud servers.

After that, in Chapter 6, based on the structure of our offloading engine, we designed a SAN model to simulate the execution of our engine. Simulations of the SAN model showed that if the offloading task needs an unknown time to migrate computation through the unreliable network connection, restarting and completing the computations locally by the mobile device can save both time and energy. Three metrics are used to evaluate and compare the performance of different thresholds, which control the moment for launching the local re-execution. The threshold is identified by multiplying the offloading execution time with a percentage. The optimal timeout fraction is found by synthetically comparing the performance under different network conditions. The advantage of launching the local re-execution at the optimal moment has been verified by the experiments with our engine in a practical application scenario.

As introduced in Chapter 7, we further proposed a congestion avoidance scheme for the mobile offloading system by using local execution. After the offloading task suffering a long delay and launching local restart to complete the task, the congestion avoidance scheme continuously handle the following offloading task several times with the local computation resource of the mobile device. A threshold is configured to control the number of local execution. After the period of local execution, the mobile client offloads the task to the server again. We designed a modular based system model with SAN to analyse the efficiency of this scheme. As PH distributions provide the best fitting result, we included them in the simulation by extending the simulation tool with a specific model. Through the simulation we found that when the whole system undertakes a heavy workload, completing offloading tasks with the local resource can effectively mitigate the congestion in the server. According to the simulation results, we proposed an optimized threshold of the number of local execution for the congestion avoidance scheme.

At last, in Chapter 8, we introduced a dynamic restart scheme to improve the performance of the mobile offloading system. In this scheme, a dynamic histogram is used to track the variation of the network quality, and the restart condition and the optimal time is estimated with the histogram. The experiment results confirm that restarting the offloading task again locally in the mobile device at the appropriate moment can reduce its completion time in some cases.

9.2 Future Work

The main goal of utilizing the restart mechanism is to quickly recover the system from the state of failure or degraded performance. Although multiple techniques and algorithms have been proposed, considering the specific environment of mobile offloading system, it is expected to investigate the following question:

(1) Since the mobile client is still a thin device with limited computation resource, the overhead of restart should be estimated before the implementation. As we have proposed in Chapter 8, the dynamic restart scheme needs to collect the data of the task completion time constantly, and then evaluates the restart benefit and calculate the optimal launching time. For the mobile client, this could be a heavy task to complete the preparation work itself. To some degree, the merit of quickly recovery could be offset by the overhead of restart. Pursuing an efficient method to accurately predict the overhead is important to evaluate the performance of restart in a more convincing manner.

(2) Another key component in our dynamic restart scheme is the distribution of the task completion time. The restart prerequisite and the optimal launching time is calculated through a proximate estimator of the probability density function. The equal width histogram used in our scheme is a simple but easily implemented method. Finding a more accurate and fast estimator for the distribution can reduce the response time of the restart scheme. The equal depth histogram is a potential option.

(3) As the distribution of the task completion time is not constant over time, a dynamic histogram is used to follow the change. A more efficient method to track the update of the histogram is worth to study. In addition, the update frequency is also a important issue in designing the tracking method. In our dynamic mechanism as introduced in Section 8.1, we check and update the histogram every time when the new sample arrives. This update rate may be too high for a relative stable distribution and it also brings a high cost of execution. Two optimized alternatives could be considered: 1). Calculating through a given algorithm to set a reasonable interval, after collecting a sufficient number of samples in this interval, the histogram is updated. 2.) According to the distance between the value of new samples and the mean of previous samples,

the update rate is changed accordingly. When the distance is wide enough, the update rate is increased, otherwise maintain the same rate. If the distances of several successive new samples stay in a low level, the update rate is decreased.

(4) The stochastic models for the mobile offloading system can also be improved by including more information. In our models, we considered all the mobile clients are all the same. As restricted by the limited computation resource, we accelerated the simulation by simply configuring all the mobile clients with identical parameters. But in the real scenario, the behaviours of these mobile clients are quite diversity. The behaviour is affected by several factors, such as the processing rate and power consumption of the micro-processor, the network quality and the interaction between the device and the user. For better understanding the impact of the diverse behaviours of the clients on the mobile offloading system, a more precise and sophisticated system model needs to be designed. This model should be able to represent the specific action of the individual client in the simulation.

(5) SAN models used in this thesis are static models. Although the parameters configured in the models are calculated from practical experiment results, they are captured by analysing the offline data. A more functional dynamic model is expected, which can predict the system performance at run time. As the continued improvement of hardware technology, for example producing processors with memristor, the performance of mobile devices and servers can be monitored and analysed at run time. That also provides the capability to update the parameters of the model with the online data. Then, the dynamic model can quickly configure these real-time parameters and produce the simulation result within a short period.

(6) For all Information Technology and Communication (ITC) systems, energy consumption is considered more and more important. As introduced in [14], the energy-related costs amount to 42% of the total budget. Reducing the energy consumption of ITC systems not only saves money for enterprises, but also helps the environment. Because the green house gases emission is positively correlated with the power consumption. For a long time, people mainly focused on the power consumption of large computer system. The energy wasted by the tiny mobile devices has been neglected. At present, more and more applications installed in the mobile device run the background service permanently. In this way, the battery can normally sustain at most one or two days. Assume that every week users have to add one more charge for the mobile device, and one charge consumes 5×10^{-3}kWh, 200 million global mobile users will consume 10^{6}kWh more energy every week. Therefore, we have to seriously consider the energy consumption when implementing the restart mechanism into the mobile offloading system.

Appendix A

Verification Experiment

In order to verify the accuracy of the fitting method introduced in Section 3.2.2, we conducted a similar experiment with the offloading test bed again. The same offloading task of recognising the image Fig. 3.1 has been repeated over a 2-hour period starting at 6pm on 22nd October 2014. Fig. A.1 shows a scatter plot of the data of 1199 samples in this period. The majority of the samples fall into the range between 3871ms and 6859ms, corresponding to the 0.02 and 0.82 quantile of all the samples. Fig. A.2 shows the histogram and density function of the samples in Fig. A.1.

The parameter results and the error measurements are shown in Table A.1. Compared with Table 3.3, the errors of fitting result in these two tables are at the same level. It proves the cluster-based fitting algorithm with PH distribution is efficient to reflect the distribution function of task completion time in the mobile offloading system. And the error of fitting results is kept in an acceptable range.

Table A.1: Hyper-Erlang parameters

T^o_{min}	3464			
Phase-Type Distribution				
m	6	5	2	4
λ*	6.1544	2.1843	0.0967	0.925324
α	0.4958	0.2932	0.0663	0.1447
Error Measurement				
$\triangle f$:	0.1934	e_1 :	0.082	

* $\times 10^{-3}$

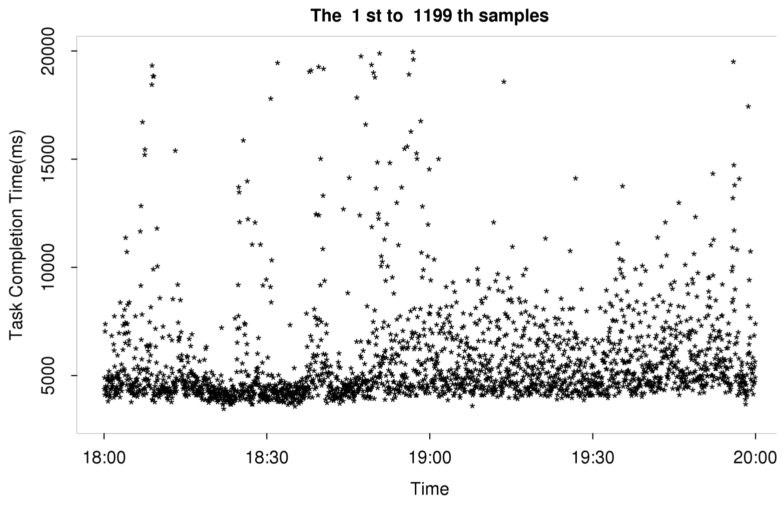


Figure A.1: Scatter plot of the OCR samples

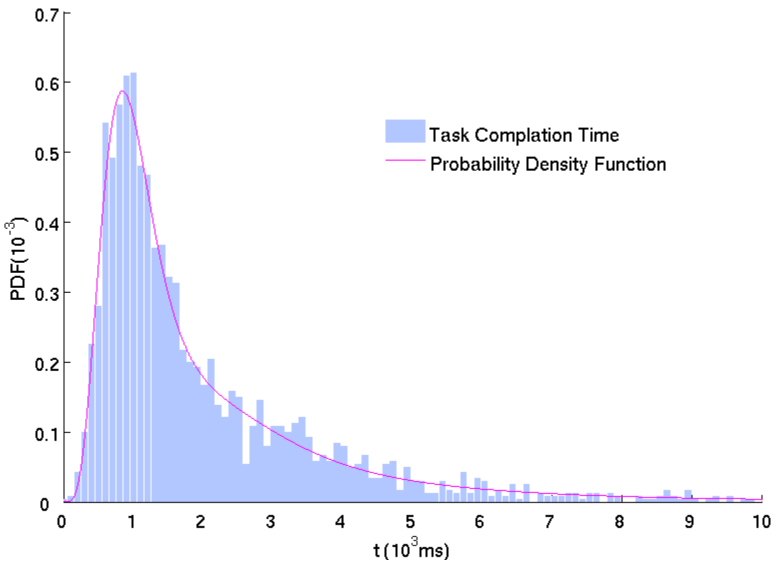


Figure A.2: Histogram and PH result of the OCR samples

Bibliography

[1] Eduroam wifi. ”https://www.eduroam.org/”.

[2] google app engine. ”https://cloud.google.com/appengine/docs”.

[3] tesseract-ocr. ”http://code.google.com/p/tesseract-ocr/”.

[4] Youdao fanyi. ”http://fanyi.youdao.com/”.

[5] Mobile cloud computing solution brief. White Paper, November 2010.

[6] AIKAT, J., HASAN, S., JEFFAY, K., AND SMITH, F. D. Discrete-approximation of measured round trip time distributions: a model for network emulation. In *First GENI Research and Educational Experiment Workshop (GREE 2012)* (2012).

[7] AIKAT, J., KAUR, J., SMITH, F. D., AND JEFFAY, K. Variability in tcp round-trip times. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement* (2003), ACM, pp. 279–284.

[8] ALTMAN, E., EL-AZOUZI, R., MENASCHE, D. S., AND XU, Y. Forever young: Aging control for smartphones in hybrid networks. *arXiv preprint arXiv:1009.4733* (2010).

[9] ANDREN, J., HILDING, M., AND VEITCH, D. Understanding end-to-end internet traffic dynamics. In *Global Telecommunications Conference, 1998. GLOBECOM 1998. The Bridge to Global Integration. IEEE* (1998), vol. 2, IEEE, pp. 1118–1122.

[10] ARMBRUST, M., FOX, A., GRIFFITH, R., JOSEPH, A. D., KATZ, R., KONWINSKI, A., LEE, G., PATTERSON, D., RABKIN, A., STOICA, I., ET AL. A view of cloud computing. *Communications of the ACM 53*, 4 (2010), 50–58.

[11] ASMUSSEN, S., FIORINI, P., LIPSKY, L., ROLSKI, T., AND SHEAHAN, R. Asymptotic behavior of total times for jobs that must start over if a failure occurs. *Mathematics of Operations Research 33*, 4 (2008), 932–944.

[12] BABCOCK, B., BABU, S., DATAR, M., MOTWANI, R., AND WIDOM, J. Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems* (2002), ACM, pp. 1–16.

[13] BALAN, R. K., SATYANARAYANAN, M., PARK, S. Y., AND OKOSHI, T. Tactics-based remote execution for mobile computing. In *Proceedings of the 1st international conference on Mobile systems, applications and services* (2003), ACM, pp. 273–286.

[14] Berl, A., Gelenbe, E., Di Girolamo, M., Giuliani, G., De Meer, H., Dang, M. Q., and Pentikousis, K. Energy-efficient cloud computing. *The computer journal 53*, 7 (2010), 1045–1051.

[15] Biaz, S., and Vaidya, N. H. Is the round-trip time correlated with the number of packets in flight? In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement* (2003), ACM, pp. 273–278.

[16] Bilmes, J. A., et al. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. *International Computer Science Institute 4*, 510 (1998), 126.

[17] Bobbio, A., and Trivedi, K. S. Computation of the distribution of the completion time when the work requirement is a ph random variable. *Stochastic Models 6*, 1 (1990), 133–150.

[18] Bohacek, S., and Rozovskii, B. A diffusion model of roundtrip time. *Computational statistics & data analysis 45*, 1 (2004), 25–50.

[19] Bolot, J.-C. Characterizing end-to-end packet delay and loss in the internet. *Journal of High Speed Networks 2*, 3 (1993), 305–323.

[20] Borella, M. S., Uludag, S., Brewster, G. B., and Sidhu, I. Self-similarity of internet packet delay. In *Communications, 1997. ICC'97 Montreal, Towards the Knowledge Millennium. 1997 IEEE International Conference on* (1997), vol. 1, IEEE, pp. 513–517.

[21] Bovy, C., Mertodimedjo, H., Hooghiemstra, G., Uijterwaal, H., and Van Mieghem, P. Analysis of end-to-end delay measurements in internet. In *Proceedings of ACM Conference on Passive and Active Leasurements (PAM), Fort Collins, Colorado, USA* (2002).

[22] Bracchi, P., Cukic, B., and Cortellessa, V. Performability modeling of mobile software systems. In *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on* (2004), IEEE, pp. 77–88.

[23] Brakmo, L. S., O'Malley, S. W., and Peterson, L. L. *TCP Vegas: New techniques for congestion detection and avoidance*, vol. 24. ACM, 1994.

[24] Brik, V., Mishra, A., and Banerjee, S. Eliminating handoff latencies in 802.11 wlans using multiple radios: applications, experience, and evaluation. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement* (2005), USENIX Association, pp. 27–27.

[25] Broido, A., and Claffy, K. Invariance of internet rtt spectrum. *Proceedings of ISMA, Leiden* (2002).

[26] Brownlee, N., and Ziedins, I. Response time distributions for global name servers. In *Proceedings of PAM 2002 Workshop* (2002).

[27] BULUT, E., AND SZYMANSKI, B. K. Wifi access point deployment for efficient mobile data offloading. *ACM SIGMOBILE Mobile Computing and Communications Review 17*, 1 (2013), 71–78.

[28] CARROLL, A., AND HEISER, G. An analysis of power consumption in a smartphone. In *USENIX annual technical conference* (2010), pp. 1–14.

[29] CÉRIN, C., COTI, C., DELORT, P., DIAZ, F., GAGNAIRE, M., GAUMER, Q., GUILLAUME, N., LOUS, J., LUBIARZ, S., RAFFAELLI, J., ET AL. Downtime statistics of current cloud solutions. *International Working Group on Cloud Computing Resiliency, Tech. Rep* (2013).

[30] CHANG, R.-S., GAO, J., GRUHN, V., HE, J., ROUSSOS, G., AND TSAI, W.-T. Mobile cloud computing research-issues, challenges and needs. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on* (2013), IEEE, pp. 442–453.

[31] CHEN, G., KANG, B.-T., KANDEMIR, M., VIJAYKRISHNAN, N., IRWIN, M. J., AND CHANDRAMOULI, R. Studying energy trade offs in offloading computation-compilation in java-enabled mobile devices. *Parallel and Distributed Systems, IEEE Transactions on 15*, 9 (2004), 795–809.

[32] CHEN, X., AND LYU, M. R. Performance and effectiveness analysis of checkpointing in mobile environments. In *Reliable Distributed Systems, 2003. Proceedings. 22nd International Symposium on* (2003), IEEE, pp. 131–140.

[33] CHICKERING, D. M., HECKERMAN, D., AND MEEK, C. A bayesian approach to learning bayesian networks with local structure. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence* (1997), Morgan Kaufmann Publishers Inc., pp. 80–89.

[34] CHUN, B.-G., IHM, S., MANIATIS, P., NAIK, M., AND PATTI, A. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems* (2011), ACM, pp. 301–314.

[35] CLARK, C., FRASER, K., HAND, S., HANSEN, J. G., JUL, E., LIMPACH, C., PRATT, I., AND WARFIELD, A. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2* (2005), USENIX Association, pp. 273–286.

[36] COCHRAN, W. G. *Sampling techniques*. John Wiley & Sons, 2007.

[37] COX, D. R. A use of complex probabilities in the theory of stochastic processes. In *Mathematical Proceedings of the Cambridge Philosophical Society* (1955), vol. 51, Cambridge Univ Press, pp. 313–319.

[38] CROVELLA, M. E., TAQQU, M. S., AND BESTAVROS, A. Heavy-tailed probability distributions in the world wide web. *A practical guide to heavy tails 1* (1998), 3–26.

[39] CUERVO, E., BALASUBRAMANIAN, A., CHO, D.-K., WOLMAN, A., SAROIU, S.,

CHANDRA, R., AND BAHL, P. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services* (2010), ACM, pp. 49–62.

[40] DABEK, F., COX, R., KAASHOEK, F., AND MORRIS, R. Vivaldi: A decentralized network coordinate system. In *ACM SIGCOMM Computer Communication Review* (2004), vol. 34, ACM, pp. 15–26.

[41] DALY, D., DEAVOURS, D. D., DOYLE, J. M., WEBSTER, P. G., AND SANDERS, W. H. Möbius: An extensible tool for performance and dependability modeling. In *Computer Performance Evaluation. Modelling Techniques and Tools*. Springer, 2000, pp. 332–336.

[42] DEAN, J., AND GHEMAWAT, S. Mapreduce: simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.

[43] DEBOOSERE, L., SIMOENS, P., DE WACHTER, J., VANKEIRSBILCK, B., DE TURCK, F., DHOEDT, B., AND DEMEESTER, P. Grid design for mobile thin client computing. *Future Generation Computer Systems 27*, 6 (2011), 681–693.

[44] DINH, H. T., LEE, C., NIYATO, D., AND WANG, P. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing 13*, 18 (2013), 1587–1611.

[45] DONJERKOVIC, D., IOANNIDIS, Y. E., AND RAMAKRISHNAN, R. Dynamic histograms: Capturing evolving data sets. In *Proceedings of the International Conference on Data Engineering* (2000), IEEE Computer Society Press; 1998, pp. 86–86.

[46] DURKEE, D. Why cloud computing will never be free. *Queue 8*, 4 (2010), 20.

[47] EGWUTUOHA, I. P., LEVY, D., SELIC, B., AND CHEN, S. A survey of fault tolerance mechanisms and checkpoint-restart implementations for high performance computing systems. *The Journal of Supercomputing 65*, 3 (2013), 1302–1326.

[48] ELTETO, T., AND MOLNAR, S. On the distribution of round-trip delays in tcp-ip networks. In *Local Computer Networks, 1999. LCN'99. Conference on* (1999), IEEE, pp. 172–181.

[49] FERNANDO, N., LOKE, S. W., AND RAHAYU, W. Mobile cloud computing: A survey. *Future Generation Computer Systems 29*, 1 (2013), 84–106.

[50] FLINN, J., PARK, S., AND SATYANARAYANAN, M. Balancing performance, energy, and quality in pervasive computing. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on* (2002), IEEE, pp. 217–226.

[51] FORD, M. D., KEEFE, K., LEMAY, E., SANDERS, W. H., AND MUEHRCKE, C. Implementing the advise security modeling formalism in möbius. In *Dependable Systems and Networks (DSN), 2013 43rd Annual IEEE/IFIP International Conference on* (2013), IEEE, pp. 1–8.

[52] FOX, A., GRIFFITH, R., JOSEPH, A., KATZ, R., KONWINSKI, A., LEE, G., PATTER-

SON, D., RABKIN, A., AND STOICA, I. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS 28* (2009), 13.

[53] FRALEIGH, C., MOON, S., LYLES, B., COTTON, C., KHAN, M., MOLL, D., ROCKELL, R., SEELY, T., AND DIOT, S. C. Packet-level traffic measurements from the sprint ip backbone. *Network, IEEE 17*, 6 (2003), 6–16.

[54] GABNER, R., HUMMEL, K. A., AND SCHWEFEL, H.-P. Modeling movable components for disruption tolerant mobile service execution. In *Cloud Computing*. Springer, 2010, pp. 231–244.

[55] GABNER, R., SCHWEFEL, H.-P., HUMMEL, K. A., AND HARING, G. Optimal model-based policies for component migration of mobile cloud services. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on* (2011), IEEE, pp. 195–202.

[56] GHOSH, R., TRIVEDI, K. S., NAIK, V. K., AND KIM, D. Interacting markov chain based hierarchical approach for cloud services.

[57] GHOSH, R., TRIVEDI, K. S., NAIK, V. K., AND KIM, D. S. End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach. In *Dependable Computing (PRDC), 2010 IEEE 16th Pacific Rim International Symposium on* (2010), IEEE, pp. 125–132.

[58] GILBERT, A. C., KOTIDIS, Y., MUTHUKRISHNAN, S., AND STRAUSS, M. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *VLDB* (2001), vol. 1, pp. 79–88.

[59] GIURGIU, I. Understanding performance modeling for modular mobile-cloud applications. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering* (2012), ACM, pp. 259–262.

[60] GIURGIU, I., RIVA, O., JURIC, D., KRIVULEV, I., AND ALONSO, G. Calling the cloud: enabling mobile phones as interfaces to cloud applications. In *Middleware 2009*. Springer, 2009, pp. 83–102.

[61] GU, X., NAHRSTEDT, K., MESSER, A., GREENBERG, I., AND MILOJICIC, D. 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* (2003), IEEE, pp. 107–114.

[62] GUMMADI, K. P., SAROIU, S., AND GRIBBLE, S. D. King: Estimating latency between arbitrary internet end hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment* (2002), ACM, pp. 5–18.

[63] GUPTA, P., AND GUPTA, S. Mobile cloud computing: the future of cloud. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*

1, 3 (2012), 134–145.

[64] Horváth, A., and Telek, M. Phfit: A general phase-type fitting tool. In *Computer Performance Evaluation: Modelling Techniques and Tools*. Springer, 2002, pp. 82–91.

[65] Horvitz, E., Ruan, Y., Gomes, C., Kautz, H., Selman, B., and Chickering, M. A bayesian approach to tackling hard computational problems. In *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence* (2001), Morgan Kaufmann Publishers Inc., pp. 235–244.

[66] Huang, D., Zhang, X., Kang, M., and Luo, J. Mobicloud: building secure cloud framework for mobile computing and communication. In *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on* (2010), Ieee, pp. 27–34.

[67] Huerta-Canepa, G., and Lee, D. A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond* (2010), ACM, p. 6.

[68] Jacobson, V. Congestion avoidance and control. In *ACM SIGCOMM Computer Communication Review* (1988), vol. 18, ACM, pp. 314–329.

[69] Jain, R. A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks. *ACM SIGCOMM Computer Communication Review 19*, 5 (1989), 56–71.

[70] Jain, R., and Chlamtac, I. The p 2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Communications of the ACM 28*, 10 (1985), 1076–1085.

[71] Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., and Towsley, D. Measurement and classification of out-of-sequence packets in a tier-1 ip backbone. *IEEE/ACM Transactions on Networking (TON) 15*, 1 (2007), 54–66.

[72] Jawerth, B., and Sweldens, W. An overview of wavelet based multiresolution analyses. *SIAM review 36*, 3 (1994), 377–412.

[73] Jelasity, M., and Montresor, A. Epidemic-style proactive aggregation in large overlay networks. In *Distributed Computing Systems, 2004. Proceedings. 24th International Conference on* (2004), IEEE, pp. 102–109.

[74] Jesus, P., Baquero, C., and Almeida, P. S. Dependability in aggregation by averaging. *arXiv preprint arXiv:1011.6596* (2010).

[75] Jiang, H., and Dovrolis, C. Passive estimation of tcp round-trip times. *ACM SIGCOMM Computer Communication Review 32*, 3 (2002), 75–88.

[76] Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., and Selman, B. Dynamic restart policies. *AAAI/IAAI 97* (2002).

[77] Kemp, R., Palmer, N., Kielmann, T., and Bal, H. Cuckoo: a computation offloading framework for smartphones. In *Mobile Computing, Applications, and Services*.

Springer, 2012, pp. 59–79.

[78] KHAN, A. R., OTHMAN, M., MADANI, S. A., AND KHAN, S. U. A survey of mobile cloud computing application models. *Communications Surveys & Tutorials, IEEE 16*, 1 (2014), 393–413.

[79] KHANNA, S., MUTHUKRISHNAN, S., AND SKIENA, S. Efficient array partitioning. In *Automata, Languages and Programming*. Springer, 1997, pp. 616–626.

[80] KLEIN, A., MANNWEILER, C., SCHNEIDER, J., AND SCHOTTEN, H. D. Access schemes for mobile cloud computing. In *Mobile Data Management (MDM), 2010 Eleventh International Conference on* (2010), IEEE, pp. 387–392.

[81] KO, S. Y., HOQUE, I., CHO, B., AND GUPTA, I. Making cloud intermediate data fault-tolerant. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 181–192.

[82] KONRAD, A., ZHAO, B. Y., JOSEPH, A. D., AND LUDWIG, R. A markov-based channel model algorithm for wireless networks. *Wireless Networks 9*, 3 (2003), 189–199.

[83] KOVACHEV, D., CAO, Y., AND KLAMMA, R. Mobile cloud computing: a comparison of application models. *arXiv preprint arXiv:1107.4940* (2011).

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

[85] KULKARNI, V., NICOLA, V., AND TRIVEDI, K. The completion time of a job on multi-mode systems. *Advances in Applied Probability* (1987), 932–954.

[86] KULKARNI, V. G., NICOLA, V. F., AND TRIVEDI, K. S. On modelling the performance and reliability of multimode computer systems. *Journal of Systems and Software 6*, 1 (1986), 175–182.

[87] KUMAR, K., LIU, J., LU, Y.-H., AND BHARGAVA, B. A survey of computation offloading for mobile systems. *Mobile Networks and Applications 18*, 1 (2013), 129–140.

[88] KUMAR, K., AND LU, Y.-H. Cloud computing for mobile users: Can offloading computation save energy. *Computer*, 4 (2010), 51–56.

[89] LI, Z., WANG, C., AND XU, R. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems* (2001), ACM, pp. 238–246.

[90] LIN, Y.-K., AND CHANG, P.-C. Maintenance reliability estimation for a cloud computing network with nodes failure. *Expert Systems with Applications 38*, 11 (2011), 14185–14189.

[91] LUBY, M., SINCLAIR, A., AND ZUCKERMAN, D. Optimal speedup of las vegas algorithms. In *Theory and Computing Systems, 1993., Proceedings of the 2nd Israel Symposium on the* (1993), IEEE, pp. 128–133.

[92] MA, Y., HAN, J. J., AND TRIVEDI, K. S. Composite performance and availability analysis of wireless communication networks. *Vehicular Technology, IEEE Transactions on 50*, 5 (2001), 1216–1223.

[93] MALHOTRA, M., AND TRIVEDI, K. S. Dependability modeling using petri-nets. *Reliability, IEEE Transactions on 44*, 3 (1995), 428–440.

[94] MARINELLI, E. E. Hyrax: cloud computing on mobile devices using mapreduce. Tech. rep., DTIC Document, 2009.

[95] MARTIN, H., MCGREGOR, A., AND CLEARY, J. Analysis of internet delay times. *Measurement 651588039* (2000).

[96] MATIAS, Y., VITTER, J. S., AND WANG, M. Wavelet-based histograms for selectivity estimation. In *ACM SIGMoD Record* (1998), vol. 27, ACM, pp. 448–459.

[97] MATIAS, Y., VITTER, J. S., AND WANG, M. Dynamic maintenance of wavelet-based histograms. In *VLDB* (2000), vol. 101, Citeseer, p. 110.

[98] MAURER, S. M., AND HUBERMAN, B. A. Restart strategies and internet congestion. *Journal of Economic Dynamics and Control 25*, 3 (2001), 641–654.

[99] MIETTINEN, A. P., AND NURMINEN, J. K. Energy efficiency of mobile clients in cloud computing. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing* (2010), USENIX Association, pp. 4–4.

[100] MISHRA, A., SHIN, M., AND ARBAUGH, W. An empirical analysis of the ieee 802.11 mac layer handoff process. *ACM SIGCOMM Computer Communication Review 33*, 2 (2003), 93–102.

[101] MUKHERJEE, A. On the dynamics and significance of low frequency components of internet load. *Technical Reports (CIS)* (1992), 300.

[102] MUPPALA, J., CIARDO, G., AND TRIVEDI, K. S. Stochastic reward nets for reliability prediction. *Communications in reliability, maintainability and serviceability 1*, 2 (1994), 9–20.

[103] OU, S., WU, Y., YANG, K., AND ZHOU, B. Performance analysis of fault-tolerant offloading systems for pervasive services in mobile wireless environments. In *Communications, 2008. ICC'08. IEEE International Conference on* (2008), IEEE, pp. 1856–1860.

[104] OU, S., YANG, K., AND LIOTTA, A. An adaptive multi-constraint partitioning algorithm for offloading in pervasive systems. In *Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on* (2006), IEEE, pp. 10–pp.

[105] OU, S., YANG, K., LIOTTA, A., AND HU, L. Performance analysis of offloading systems in mobile wireless environments. In *Communications, 2007. ICC'07. IEEE International Conference on* (2007), IEEE, pp. 1821–1826.

[106] PARK, J., YU, H., CHUNG, K., AND LEE, E. Markov chain based monitoring service

for fault tolerance in mobile cloud computing. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on* (2011), IEEE, pp. 520–525.

[107] POOSALA, V., HAAS, P. J., IOANNIDIS, Y. E., AND SHEKITA, E. J. Improved histograms for selectivity estimation of range predicates. In *ACM SIGMOD Record* (1996), vol. 25, ACM, pp. 294–305.

[108] POWERS, R. A. Batteries for low power electronics. *Proceedings of the IEEE 83*, 4 (1995), 687–693.

[109] QI, H., AND GANI, A. Research on mobile cloud computing: Review, trend and perspectives. In *Digital Information and Communication Technology and it's Applications (DICTAP), 2012 Second International Conference on* (2012), ieee, pp. 195–202.

[110] QIAN, H., AND ANDRESEN, D. Emerald: Enhance scientific workflow performance with computation offloading to the cloud. In *Computer and Information Science (ICIS), 2015 IEEE/ACIS 14th International Conference on* (2015), IEEE, pp. 443–448.

[111] QIAN, H., AND ANDRESEN, D. An energy-saving task scheduler for mobile devices. In *Computer and Information Science (ICIS), 2015 IEEE/ACIS 14th International Conference on* (2015), IEEE, pp. 423–430.

[112] QIAN, H., AND ANDRESEN, D. Jade: Reducing energy consumption of android app. *the International Journal of Networked and Distributed Computing (IJNDC), Atlantis press* (2015).

[113] QIAN, H., AND ANDRESEN, D. Reducing mobile device energy consumption with computation offloading. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on* (2015), IEEE, pp. 1–8.

[114] RAMANI, I., AND SAVAGE, S. Syncscan: practical fast handoff for 802.11 infrastructure networks. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE* (2005), vol. 1, IEEE, pp. 675–684.

[115] REINECKE, P., KRAUSS, T., AND WOLTER, K. Cluster-based fitting of phase-type distributions to empirical data. *Computers & Mathematics with Applications 64*, 12 (2012), 3840–3851.

[116] REINECKE, P., KRAUSS, T., AND WOLTER, K. Hyperstar: Phase-type fitting made easy. In *QEST* (2012), pp. 201–202.

[117] REINECKE, P., VAN MOORSEL, A., AND WOLTER, K. A measurement study of the interplay between application level restart and transport protocol. In *Service Availability*. Springer, 2005, pp. 86–100.

[118] REINECKE, P., AND WOLTER, K. Adaptivity metric and performance for restart strategies in web services reliable messaging. In *Proceedings of the 7th international workshop*

on Software and performance (2008), ACM, pp. 201–212.

[119] Ruan, Y., Horvitz, E., and Kautz, H. Restart policies with dependence among runs: A dynamic programming approach. In *Principles and Practice of Constraint Programming-CP 2002* (2002), Springer, pp. 573–586.

[120] Salfner, F., and Wolter, K. Analysis of service availability for time-triggered rejuvenation policies. *Journal of Systems and Software 83*, 9 (2010), 1579–1590.

[121] Sanders, W. H., and Meyer, J. F. Stochastic activity networks: Formal definitions and concepts. In *Lectures on Formal Methods and PerformanceAnalysis*. Springer, 2001, pp. 315–343.

[122] Satyanarayanan, M. Mobile computing. *Computer 26*, 9 (1993), 81–82.

[123] Satyanarayanan, M. Pervasive computing: Vision and challenges. *Personal Communications, IEEE 8*, 4 (2001), 10–17.

[124] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE 8*, 4 (2009), 14–23.

[125] Shakkottai, S., Srikant, R., Brownlee, N., Broido, A., et al. The rtt distribution of tcp flows in the internet and its impact on tcp-based flow control.

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

[127] Shieh, Y.-B., Ghosal, D., Chintamaneni, P. R., and Tripathi, S. K. Application of petri net models for the evaluation of fault-tolerant techniques in distributed systems. In *Distributed Computing Systems, 1989., 9th International Conference on* (1989), IEEE, pp. 151–159.

[128] Shrivastava, N., Buragohain, C., Agrawal, D., and Suri, S. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems* (2004), ACM, pp. 239–249.

[129] Shvachko, K., Kuang, H., Radia, S., and Chansler, R. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on* (2010), IEEE, pp. 1–10.

[130] Smith, R. An overview of the tesseract ocr engine. In *ICDAR* (2007), vol. 7, pp. 629–633.

[131] Srirangarajan, S., Allen, M., Preis, A., Iqbal, M., Lim, H. B., and Whittle, A. J. Wavelet-based burst event detection and localization in water distribution systems. *Journal of Signal Processing Systems 72*, 1 (2013), 1–16.

[132] Tasche, D. Conditional expectation as quantile derivative. *arXiv preprint math/0104190* (2001).

[133] TELEK, M., AND HEINDL, A. Matching moments for acyclic discrete and continuous phase-type distributions of second order.

[134] THAPER, N., GUHA, S., INDYK, P., AND KOUDAS, N. Dynamic multidimensional histograms. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data* (2002), ACM, pp. 428–439.

[135] THOMPSON, K., MILLER, G. J., AND WILDER, R. Wide-area internet traffic patterns and characteristics. *Network, iEEE 11*, 6 (1997), 10–23.

[136] THUMMLER, A., BUCHHOLZ, P., AND TELEK, M. A novel approach for phase-type fitting with the em algorithm. *Dependable and Secure Computing, IEEE Transactions on 3*, 3 (2006), 245–258.

[137] TRIVEDI, K., AND MA, X. Performability analysis of wireless cellular networks. In *Proceedings of Symposium on Performance Evaluation of Computer and Telecommunication System (SPECTS 2002), San Diego, USA* (2002).

[138] TRIVEDI, K. S., MA, X., AND DHARMARAJA, S. Performability modelling of wireless communication systems. *International journal of communication systems 16*, 6 (2003), 561–577.

[139] TRIVEDI, K. S., MALHOTRA, M., AND FRICKS, R. M. Markov reward approach to performability and reliability analysis. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 1994., MASCOTS'94., Proceedings of the Second International Workshop on* (1994), IEEE, pp. 7–11.

[140] VAN MOORSEL, A. P., AND WOLTER, K. Analysis and algorithms for restart. In *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the* (2004), IEEE, pp. 195–204.

[141] VAN MOORSEL, A. P., AND WOLTER, K. Meeting deadlines through restart. In *MMB* (2004), pp. 155–160.

[142] VAN MOORSEL, A. P., AND WOLTER, K. Analysis of restart mechanisms in software systems. *Software Engineering, IEEE Transactions on 32*, 8 (2006), 547–558.

[143] VAN NIEUWPOORT, R. V., MAASSEN, J., WRZESIŃSKA, G., HOFMAN, R. F., JACOBS, C. J., KIELMANN, T., AND BAL, H. E. Ibis: a flexible and efficient java-based grid programming environment. *Concurrency and Computation: Practice and Experience 17*, 7-8 (2005), 1079–1107.

[144] VITTER, J. S., AND WANG, M. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *ACM SIGMOD Record* (1999), vol. 28, ACM, pp. 193–204.

[145] VITTER, J. S., WANG, M., AND IYER, B. Data cube approximation and histograms via wavelets. In *Proceedings of the seventh international conference on Information and knowledge management* (1998), ACM, pp. 96–104.

[146] WANG, B., SECHILARIU, M., AND LOCMENT, F. Power flow petri net modelling for building integrated multi-source power system with smart grid interaction. *Mathematics and Computers in Simulation 91* (2013), 119–133.

[147] WANG, C., AND LI, Z. Parametric analysis for adaptive computation offloading. In *ACM SIGPLAN Notices* (2004), vol. 39, ACM, pp. 119–130.

[148] WANG, D., XIE, W., AND TRIVEDI, K. S. Performability analysis of clustered systems with rejuvenation under varying workload. *Performance Evaluation 64*, 3 (2007), 247–265.

[149] WANG, J., LIU, J., AND SHE, C. Segment-based adaptive hyper-erlang model for long-tailed network traffic approximation. *The Journal of Supercomputing 45*, 3 (2008), 296–312.

[150] WANG, Q., AND WOLTER, K. Reducing task completion time in mobile offloading systems through online adaptive local restart. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering* (2015), ACM, pp. 3–13.

[151] WANG, Z., AND CROWCROFT, J. A new congestion control scheme: Slow start and search (tri-s). *ACM SIGCOMM Computer Communication Review 21*, 1 (1991), 32–43.

[152] XIAN, C., LU, Y.-H., AND LI, Z. Adaptive computation offloading for energy conservation on battery-powered systems. In *Parallel and Distributed Systems, 2007 International Conference on* (2007), vol. 2, IEEE, pp. 1–8.

[153] YANG, K., OU, S., AND CHEN, H.-H. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. *Communications Magazine, IEEE 46*, 1 (2008), 56–63.

[154] ZHANG, Q., AND WANG, W. A fast algorithm for approximate quantiles in high speed data streams. In *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on* (2007), IEEE, pp. 29–29.

[155] ZHANG, Y., BRESLAU, L., PAXSON, V., AND SHENKER, S. On the characteristics and origins of internet flow rates. In *ACM SIGCOMM Computer Communication Review* (2002), vol. 32, ACM, pp. 309–322.

[156] ZHANG, Y., AND DUFFIELD, N. On the constancy of internet path properties. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement* (2001), ACM, pp. 197–211.

List of Figures

List of Tables

About the Author

Qiushi Wang received his bachelor and master degrees from Beijing University of Posts and Telecommunications in 2008 and 2011 respectively. Supported by China Scholarship Council (2011-2015), he joined in Computer System and Telematics group of Freie Universität Berlin, supervised by Prof. Dr. Katinka Wolter since 2011. His research interests include mobile offloading system, stochastic modelling and restart algorithm.