TU Dortmund

Master Thesis

Design and Implementation of Computation Offloading Mechanism in Real-Time System

Author:
Huan Tian

Supervisor:
Prof. Dr. Jian-Jia Chen

July 2015

TU DORTMUND

Abstract

Automation & Robotics
Department of Information

Master of Science

Offloading Mechanism Realization in Real Time Operating System

by Huan TIAN

Computation offloading is a widely used method to improve the performance of embedded systems by offloading some computation-intensive tasks to some powerful components. In this thesis, we propose a computation offloading mechanism in real-time systems. When the task model is complicated, a HEU offloading decision algorithm and a priority-based HEU offloading decision algorithm have been proposed based on the approximated demand bound functions. We use this computation offloading mechanism in virus detection. In the case study of virus detection, we show the effectiveness of our mechanism. By some simulation results, we compare two different offloading decision algorithms.

Contents

List of Figures

List of Tables

Abbreviations

RTOS	**R**eal **T**ime **O**perating **S**ystem
WCET	**W**orst **C**ase **E**xecution **T**ime
EDF	**E**aliest **D**eadline **F**irst
DBF	**D**emand **B**ound **F**unction
ADBF	**A**pproximated **D**emand **B**ound **F**unction
MMKP	**M**ultiple **M**ultidimensional **K**napsack **P**roblem

Chapter 1

Introduction

1.1 Cyber-Physical System

Cyber-Physical Systems (CPS) are integrations of computation, networking, and computational elements, which can enable the physical world to connect with the virtual world. Today, Cyber-physical systems can be found in many areas such as aerospace, automotive driving, civil infrastructure, healthcare, manufacturing, transportation and so on. The economic and societal potential of Cyber-Physical Systems is vastly greater. More investments are being made worldwide to develop the technology in recent years.

Using sensors, the Cyber-Physical system can monitor and collect data from the environment. The systems are normally connected with the internet globally. In the design of Cyber-Physical systems, one of the challenges is about an increasing number of data processing applications nowadays. With the increasing data sets, data analysis, search, sharing, storage, transfer, visualization and privacy make the design of Cyber-Physical systems much more difficult. Effectively extracting important information from data and predicting the potential behavior make an embedded system much more costly.

Data-intensive applications are commonly ultilized in current Cyber-Physical Systems. For instance, virus can be really jeopardizable and cause catastrophic damage. Virus detection technology seems essential and imperative under such circumstance. To detect a virus which is a nano-sized object, a large number of data processing and analysis are needed. An efficient, accurate and propagative detection technology should be designed

more reliable and user friendly. How to devise such a kind of virus-detection device is still an open problem now.

Moreover, with the growing number of data sets, how to effectively work with big data has become a necessary problem. Considering about these requirements, one simple embedded device should be used as a powerful platform to finish many complicated applications. Such an embedded device can be easily transplanted to many intelligent devices such as smart phone, laptop or even simple micro-controller based devices. Some of these application are also data intensive applications. For a normal embedded system, one big problem, however, for most complicated applications, is resource constrained problem. For example, for an image processing application, more and more complicated algorithms, such as machine learning based algorithms, have been employed. Such scale computing requires high performance computing platform, which seems difficult for computing components of embedded system.

Over the past several years, many solutions have been studied to address the problem. Some commonly used methods include improve the computing ability and ultilize more parallel processing elements. However, these methods need to put much more hardware or software efforts. Compared with these methods, one simple and easier implemented solution is computation offloading.

1.2 Computation Offloading Techniques

Many embedded systems have limited resources, such as computing ability, battery life, network bandwidth, storage capacity and so on. Computation offloading techniques can effectively alleviate these constraints by sending complicated computation to some resourceful components and receiving the results to finish the whole computation. Computation offloading techniques are typically used to augment the computational capability of some resource constrained device. Moreover, computation offloading techniques are quite different from the traditional client-server architectures, where the client always migrates computation to the serve[4]. Moreover, computation offloading techniques are also different from load balancing strategies in multiprocessor systems or grid computing systems[5].

Offloading some complicated computations from resource-constrained devices to some powerful components have many benefits, such as saving energy consumption and improving performance[6],[7].

There are a significant amount of research papers about computation offloading techniques. In their works, most of these papers are focused on following questions:

Is it necessary to implement such mechanism? Measurements should be carried out to find out whether energy or performance can be saved and improved due to the implementation. If the resource consumption does not change or slightly change, then it seems not worth realizing such mechanism given consideration of energy and financial cost for realization of mechanism.

When and where to implement such mechanism? A criteria, offloading decision, should be determined in order to decide which parts of program or algorithms should be uploaded. In such way, the mechanism could be feasibly and efficiently applied. When dealing with RTOS, other problems such as timing critical problem should also be taken into account while catastrophically outcomes may occur due to deadline missing. Then more sophisticated methods should be proposed to address the issue.

However, in previous paper, seldom works consider the computation offloading techniques in real-time embedded systems.

1.3 Computation Offloading in Real Time System

The real-time system is a system which can control an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time. Hardware and software systems must subject to the timing constraint. In real-time systems, each application must finish its computation within specified time constraints, often referred to as deadlines. Such systems are often used for many mission critical areas such as for vehicles, robotics, aerospace and so on. In these areas, missing deadline for some applications will lead to disaster sequences.

Applying computation offloading mechanism in real-time embedded systems are meaningful. For many applications, in order to get precise results or decision making, more sophisticated algorithms must be applied. For example, machine learning algorithms

are difficult for some mobile embedded systems to accomplish some tasks. Let us take an image based navigation robotic for example. The general principle behind vehicle robotics is trying to let the robotics make the decision itself based on the chosen features yielded from images recorded by a camera device. The procedures or algorithms dealing with images and features could be really time consuming due to the complexity and computation scale. So the ideal solution will be uploading the images or selected parts of programmings along with inputs to remote server which has more powerful CPU or GPU, with which the calculation will be more quickly accomplished. So it seems that with offloading mechanism, more complicated algorithms can be carried out on embedded devices.

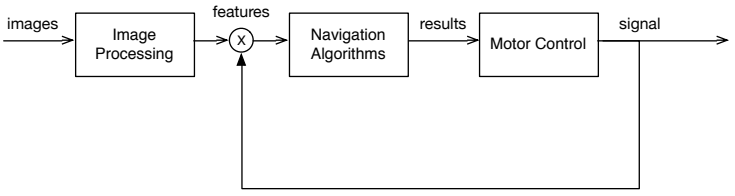


FIGURE 1.1: basic control structure of vision based navigation robotics[1]

Fig. 1.1 illustrates one of the basic control schemes of vision-based navigation robot. From the figure it can be noticed that the control strategy can be regarded as an on-line dynamic processing which not only demands accuracy, sometimes, but also time-effectiveness. When the robot tries to avoid obstacles, for example, it is clear that efficient results should be required immediately such that the robot can make a turning in time. Otherwise collisions may happen. So it also can be seen as a real time operating system in some senses.

In the design of real-time embedded systems, deadline can be seen as an essential parameter because missing deadline can lead to catastrophically results. So during the design of real-time embedded system, how to implement computation offloading mechanism is still difficult problem. How to find an effective computation offloading mechanism is also our focus. Moreover, in real-time systems, computation offloading often will lead to some self-suspension time for each real-time task. Such a self-suspension time will lead to some bad effects for the scheduling analysis. How to guarantee all tasks with self-suspension time is a hot topic is real-time systems. Based on the scheduling analysis, we need to decide the offloading decisions such as which parts of programs should be

offloaded and which parts of programs should be executed locally. Such a problem is known to be NP-hard problem [8].

1.4 Contributions

Here, offloading mechanism is realized with real time system. During which, some problems forementioned will be addressed with algorithms such as EDF, Approximated Demand Bound Function (ADBF) which will be introduced later. In order to realize such mechanism, some tasks models must be set up. In the end, to illustrate the idea, a real project – virus detection project is introduced to realize the mechanism.

In chapter 2, basic real time task concepts are introduced such as tasks modeling, tasks scheduling algorithms. These will be used laster for scheduling model of multiple tasks offloading mechanism. Later, related work about offloading mechanism is introduced, one self-suspension problem is also introduced.

In chapter 3, the general offloading mechanism model is proposed, one task model or multiple tasks model. Procedures about addressing such tasks scheduling problem are presented. An approximation demand bound function (ADBF) is adopted to simply the scheduling procedure. In the next few sections some algorithms are adopted. Among which, one classification based ADBF method is discussed to reduce the approximation errors. Also due to the NP-hard scheduling problem, HEU algorithm is used to yield the optimal solution of this MMKP issue. In the end one priority based algorithm is introduced to accelerate the candidates iteration procedures.

In case study chapter, one virus detection project is introduced. The first section concentrates the introduction of the project, general methods of the detection, basic techniques and devices like such as PAMONO sensor, CCD camera. Then the basic principle and processing procedures are presented. Next, a portable detection device concept is proposed which is the main aim of the project, specific software design, related tools are presented. In the end, one demo implementation of client is illustrated which processes pre-processing procedures on the local embedded devices. Specific processing procedures such as image cutting, de-noising and image skipping are discussed in experiment section.

In the last chapter, tasks scheduling solution is given through one UI interface. In the first, one general model is set up and one general UI interface is designed to address the scheduling problem. The final interface design is given in tool build section. In next two sections, parameters and penalty as well as priority are discussed. The determination method and consideration is presented. Results and conclusion can be found in the final section.

Chapter 2

Backgrounds

In this chapter, we mainly give some backgrounds which are related to this dissertation. First, we begin with an overview of real-time scheduling, including task model, common scheduling algorithms and scheduling analysis. Second, we introduce some related works including offloading decision algorithms and self-suspension time analysis in real-time systems.

2.1 Overview of Real Time Scheduling

In this section, some basic concepts about modeling tasks in real time system are introduced which will be adopted in latter sections.

2.1.1 Real-time task model

In real-time systems, most workloads require the execution at recurring intervals. One of the simplest and most common used task model is the periodic task model, which was first introduced by Liu and Layland. In this model, each task τ_i is specified by a worst-case per-job execution cost C_i and a period P_i. The utilization of the task τ_i is defined as $\frac{C_i}{T_i}$. Each task release jobs periodically. The release time of one job from τ_i is denoted as r_i. The absolute deadline of a job from τ_i is $r_i + D_i$, where D_i is the relative deadline of τ_i. Normally, the arrival time a_i of one job from τ_i is defined as the time when a task comes out from system queen and is ready for execution. Computation time

C_i, unlike task period, is the real execution time of task which begins at the starting time S_i, and ends at the finishing time F_i. Absolute deadline d_i indicates the time when system has already completed the current task and is ready to process the coming task.

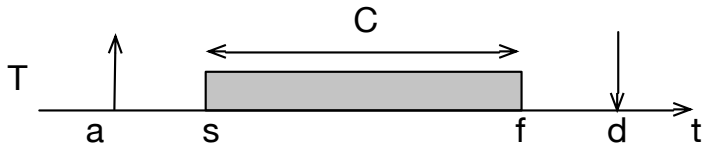


FIGURE 2.1: basic concepts for task model.[2]

Fig. 2.1 illustrates the typical parameters of a real-time task in detail. In this thesis, when the system has several real-times at the same time, how to schedule these tasks is a non-trivial problem. Figure 2.1 illustrates

Unlike the periodic task model, another task model is the sporadic task model. In this model, T_i specifies the minimum separation between two successive released jobs, not the exact separation in the periodic task model.We present the sporadic task model here, but the periodic task model is assumed here unless stated otherwise.

2.1.2 Real-time scheduling algorithms

In this part, we briefly introduce fixed priority scheduling and dynamic priority scheduling algorithm as follows:

Fixed priority Scheduling is one class of scheduling algorithm in real-time systems. In real-time systems, the priority of each task is used to schedule different released jobs. In fixed priority scheduling, each job is assigned a fixed priority via some policy. Contention for resources is resolved in favor of the job with the higher priority that is ready to run.

In fixed priority scheduling, all the jobs of a task have the same priority. A task, which is a process or thread, has a specified priority. For instance, assume all tasks are numbered and task τ_i has priority i. The higher value of i denotes a higher priority and the lower value of i denotes a lower priority. One of the typical task-level priority assignments is known as rate-monotonic scheduling. In rate-monotonic scheduling, the priorities are

assigned on the basis of the task periods. The shorter the task period is, the higher priority the job is.

Dynamic priority Scheduling is another class of scheduling algorithm. Task priorities are assigned to individual jobs, not tasks like fixed priority scheduling. One of the most used algorithms is the Earliest Deadline First(EDF) according to the priority which is inversely proportional to the absolute deadlines of the active jobs. EDF scheduling algorithm has been proved as the optimal preemptive scheduling algorithms. That means, if these exists a feasible schedule for a task set, the the schedule produced by EDF is also a feasible schedule. EDF scheduling algorithm can be extended to some more general cases in real-time systems.

2.1.3 Schedulability analysis

In real-time systems, a task set is feasible when there exists some scheduling algorithm that can schedule all possible released jobs and all these jobs generated by the task set don't miss any deadlines. A task is referred to as schedulable according to a given scheduling algorithm if its worst-case response time under that scheduling algorithm is less than or equal to its deadline. Similarly, a task set is referred to as schedulable according to a given scheduling algorithm if all of its tasks are schedulable. Schedulability analysis is used to predict temporal behavior which determine whether all tasks can meet its timing constraint during the run time. Such analysis are constrained by several factors. Sufficient and necessary analysis are ideal and intractable. The complexity of such analysis is NP-hard for many different task models. So given us a set of real-time tasks, it is needed to analysis these tasks all these tasks can be schedulable by some real-time scheduling algorithms.

For **Fixed priority scheduling** algorithm, in 1973, Liu and Layland [9] propose a foundational and influential work in fixed-priority real-time scheduling theory. They assume all tasks are periodic and released at the beginning of the period. The relative deadline of each task is equal to its period. All tasks are independent, without resource or precedence relationships. Each task has a specified worst case execution time, which is less than or equal to its period. All tasks are pre-emptible on one processor.

Liu and Layland propose the Critical Instant Theorem for the feasibility of fixed priority scheduling algorithm. For a given task, the critical instant is the release time when the response time is maximized. From the theorem, for a given set of periodic tasks with fixed priority, a critical instance for a task occurs when it is simultaneously released with all higher priority tasks. For task τ_i, all higher priority tasks releasing at 0 can create the hardest situation, with the maximum response time of τ_i.

The critical instant theorem provides an obvious necessary and sufficient test for feasibility of fixed priority scheduling algorithm. That is to say, in order to find the worst case execution conditions, we can assume that all tasks initially released together. The task set is feasible if and only if all tasks can finish before its first deadline. We only need to consider points which correspond to task deadlines and release times.

Based on the concept of critical instant, when a set of tasks are assigned priorities according to the rate-monotonic policy, a sufficient utilization-based condition can be used for feasibility. Assume a set of n periodic tasks is feasible, if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}}) - 1$$

Although the Liu and Layland schedulability analysis is simple, the feasibility condition is sufficient but not necessary. It needs all tasks have equal relative deadline and period. Moreover, task priorities must be assigned according to the rate-monotonic policy. An improved schedulability analysis has been proved as follows. If a set of tasks is released together at 0, the i highest priority task will complete its first execution within its deadline if there is a time $0 < t < T_i$ such that the demand on the processor, W_i, of the i highest priority tasks is less than or equal to t, that is [10],

$$W_i(t) = \sum_{j=1}^{i} \lceil \frac{t}{T_j} \rceil C_j \leq t$$

The only values of t which need to be checked are the multiples of the task periods between 0 to T_i. This test can be used to test arbitrary fixed priority orderings.

Moreover, fixed-priority response-time analysis has also been proposed. The algorithm computes the worst-case response time R_i of task τ_i with the following recursive equation:

$$R_i = C_i + \sum_{j=1}^{i} \lceil \frac{R_i}{T_j} \rceil C_j$$

Only a limited points are needed to check for the system feasibility.

For **Dynamic Priority Scheduling** algorithm, task priorities are based on each job. The Earliest Deadline First (EDF) is one of the most widely used algorithm. If execution deadlines are equal to their periods for all tasks, analysis of schedulability analysis under EDF algorithm can be simply carried out by guaranteeing that the utilization of all tasks is no more than one. This set of n periodic tasks is schedulable by the EDF algorithm, if and only if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

When task deadlines are not equal to periods, scheduability analysis becomes much more complicated. One popular method is demand bound function (DBF) analysis. Demand bound function is defined as the maximum amount of resource demand of current jobs within a time period t. The demand bound function can be defined as follow:

$$dbf(\tau_i, t) = \lfloor \frac{t + T_i - D_i}{T_i} \rfloor C_i$$

A set of tasks can be scheduled by EDF scheduling algorithm if and only if demand bound functions of all tasks must be no more than 1. That is to say

$$\forall t \geq 0, \sum_{i=1}^{n} dbf(\tau_i, t) \leq t, \tag{2.1}$$

2.2 Related Works

2.2.1 Offloading Decision in Real-time Systems

Thanks to the recent advances in mobile and wireless technologies, mobile computing devices have become very essential in our daily life. Devices such as smart phones, mobile robots and wearable computers can be seen everywhere. Multiple tasks can be

executed simultaneously on such devices, which include voice and image recognition, navigation, video processing, etc.

However, such devices can be resource limited due to constrains such as computation capabilities, memory capacity and battery life. Hence, for those tasks of complicated algorithms and time consuming computations, such devices may not be efficient to finish the execution of all the tasks in time. In this case, the results may become useless or even harmful to the system due to the deadline missing. One solution is to use the computation offloading, where the mobile device (i.e., the resource-constrained device) offloads the computation intensive tasks to a powerful remote processing unit. The remote processing unit executes the offloaded tasks and returns the results back to the mobile device. Such embedded system and powerful remote processing unit construct simple client-server system.

Many studies about such offloading mechanism has been carried on for a while. Paper [11], [12], [13] determine the offloading decision based on comparison between the time consumed during the local execution and the time consumed during the offloading for each task. While in [14], [15], [16], [17], [18] graph partitioning method have been adopted to solve the computation offloading problem. Task scheduling or the server model, however, have not been considered jet in forementioned approaches which only focus on the offloading decision. Also, most of them either do not consider the timing satisfaction requirement for real-time properties, or use pessimistic offloading mechanism for deciding whether a task can be offloaded or not [19]. Paper [20] uses total bandwidth server (TBS)[21], [22] on the server side to provide resource reservation for the offloaded tasks. While paper [23] focuses on the client who finds a feasible schedule for the tasks such that the required utilization from the server is minimized, in order to avoid wasting the resources of the server.

While in this part, one specific model of offloading mechanism is built up to alleviate the computational pressure of local devices due to forementioned constrains. In the model, multiple tasks are executed simultaneously on the sever while offloading mechanism is applied for all tasks. Considering multiple versions with respect to each task due to alternative offloading decisions, such tasks scheduling problem can be considered as a NP-hard problem. Therefore, in this thesis, studies about how to schedule such model will be carried on.

2.2.2 Self-suspension Time analysis

During the offloading, suspension delays may occur as mentioned. Suspension delay can be caused due to tasks block to access shared resources or interact with external devices such as I/O. Such delays can be quite lengthy (e.g., 15ms for a disk read), which can cause really negatively impact on schedulability in real-time systems. Studies have been shown that precisely analyzing hard real-time (HRT) systems with suspensions is difficult, even for very restricted suspending task models on uniprocessors[8].

Two major categories of techniques have been adopted to analyze suspensions, suspension-oblivious v.s. suspension-aware analysis. Under suspension-oblivious analysis (which is perhaps the most commonly used approach), suspensions are simply integrated into per-task worst-case execution time requirements. However, this approach clearly yields utilization loss. The alternative is to employ suspension-aware analysis,where suspensions are explicitly considered in the task model and resulting schedulability analysis. On a uniprocessor,suspension-aware analysis techniques that are correct in a sufficiency sense have been proposed [24], [25], [26], [27]. However, these analysis can be quite pessimistic in many cases. Indeed, suspensions are hard to analyze because they may leave the processor idle and it is impossible to predict when suspensions may occur in the runtime schedule. This causes pessimism in the analysis because we have to assume that all suspending tasks suspend concurrently at any idle time instant. For scheduling soft real-time (with guaranteed bounded response times) suspending task systems on multiprocessor, studies can be found in[13], [28].

For the HRT case, besides the suspension-oblivious approach of treating all suspensions as computation, several schedulability tests have been presented for analyzing periodic tasks that may suspend at most once on a uniprocessor[24],[25],[26],[27]. Unfortunately, these tests are rather pessimistic as their techniques involve straightforward execution control mechanisms, which modify task deadlines. For example, a suspending task that suspends once can be divided into two subtasks with appropriately shorted deadlines and modified release times. Such techniques inevitably suffer from significant capacity loss due to the artificial shortening of deadlines.

In this thesis, models are built up based on offloading mechanism, which can cause self-suspension. Solutions, however, will be found in order to avoid such time unreliable delay. Offloading model can be set up without waiting for any receive from remote

server. In such way, system of local can regard the offloading tasks as simple normal tasks. Details about task modeling can be found in next chapter.

Chapter 3

Offloading Decisions Algorithms

In previous chapter, scheduling problem has been discussed when dealing with implementation of offloading mechanism in real time system. In this chapter, more details will be discussed. Models about offloading mechanism will be set up and presented. Algorithms about scheduling such models will also be introduced, which is the main concern of this thesis.

3.1 Software Architecture

First, a general architecture of computation offloading mechanism will be proposed. Offloading mechanism, as mentioned, can be adopted in many different areas. Discard of different specific applications, the architecture of offloading mechanism can be extract and depicted as in Fig. 3.1.

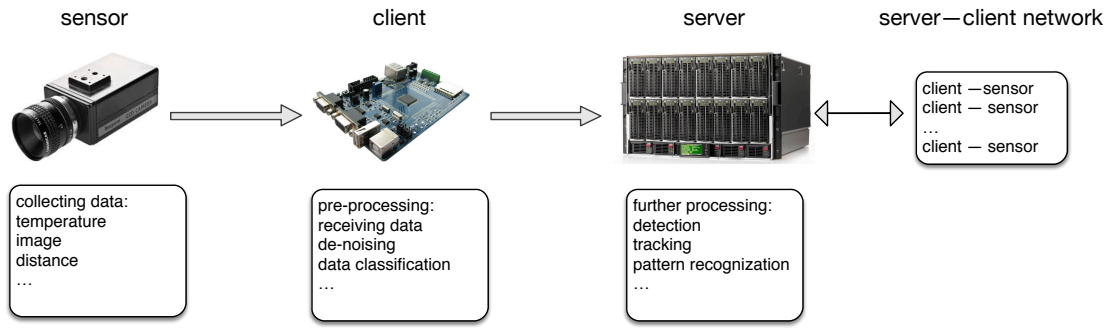


FIGURE 3.1: offloading application system structure

This general architecture has the following basic components:

- **Sensors**: The sensor part is to sense (or detect) some events or changes from environment. According to different requirements, different kinds of sensors can be adopted to collect environment information, such as temperature and humidity sensors, velocity sensors, chemical sensors, biosensor and so on. Some cameras, which can capture images from environments, can also be deemed as one type of sensors.

- **Embedded Devices**: Embedded devices are responsible to collect data from sensors and to do some calculations. However, due to the limited computing capacity, the embedded system can only do some basic applications and some simple preprocessing for future works. For some complicated applications. The embedded systems can also send some complication computations to other powerful components.

- **Servers**: The server part can be any powerful components, which are suitable for some data intensive applications. These components can be Graphics Processor Unit(GPU), Field Programmable Gate Array(FPGA), a server or a data center. The server part can handle more complicated applications, which can alleviate computations of embedded devices. For some components, such as servers or data centers, data storage is also very larger compared with embedded systems.

- **Networks**: The connect between embedded devices and servers are is mainly based on networks. Such network can be WIFI, zigbee, or fiber optic network depending to requirements.

In this computation offloading mechanism, there are two different offloading conditions. The first one is computation results can come back. That is to say, computation results from servers are important for embedded devices. Sometimes these results are used for further analysis in embedded devices. The other one is computation results can not come back. In some conditions, due to limited computation ability and storage capacity, embedded systems do not need the analysis results from servers. In the first condition, when computations can come back, the whole task can be divided into three parts. As in Fig. 3.2, subtask A and B are serial parts of the original real-time task. There exists an idle time between subtask A and subtask B. This idle time is called self-suspension time in real-time systems. Such a self-suspension time make the system not easy to be scheduled. Moreover, the analysis with such self-suspension time is also much more

difficult. In order to schedule this task wish self-suspension time, we propose to assign different relative deadline for each sub task. When each sub task A and B has different release time and relative deadline, then traditional scheduling algorithms can be used in real-time systems.

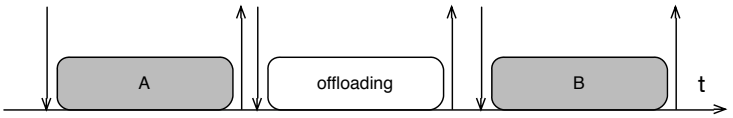


FIGURE 3.2: illustration of offloading mechanism model

For a given application, the self-suspension time means which part can be offloading to servers. Normally, in the design of embedded systems, we have different solutions of offload different application. Different offloading solutions will lead to different system performances.

Moreover, in the design of such a system, we need to consider an offloading decision problem. This part is essential for the system designs as the offloading decision will influent the whole system. In this problem, we need to consider the following two questions:

- How to model the offloading problem? In the offloading procedure, we need to guarantee all tasks are feasible in real-time systems. Normally, the connection between embedded devices and servers is just communication, such as socket programming. It can be easily realized with programming language. In the socket programming, no matter TCP or UDP protocols, each connection between the embedded devices and servers has standard procedures. When the connection between embedded systems and servers is established, then the data transfer can start by the send-receive model in socket programming. In this procedure, we need to analyze the system feasibility and check all tasks can be scheduled by a specified scheduling algorithm. Hence, we need to exactly model the whole offloading procedure and analyze the scheduling algorithm in embedded devices.

- How to do the offloading decision? When and which parts of the tasks should be uploaded? For any given application, obviously, there are different solutions to offloading computations, including different offloading time, different offloading

parts and so on. Different offloading solutions have different effects and the offloading decisions will have influences. Let us take an example, different the object detection procedure, there are several offloading solutions such as different offloading images. And the offloading times may be also at different points. Different solutions will have different detected accuracy. Hence, the problem how to do the offloading is also a non-trivial problem.

In order to solve the following two problems, we propose the following system model as follows:

3.2 System Model

In this part, when several periodic tasks released simultaneously on a uniprocessor, a schedulability test should be made exactly. Suppose there are n tasks independent periodic tasks, τ_1,...,τ_n, scheduled by the EDF scheduling algorithm. For a task τ_i, we can have different offloading methods to offload this task. However, different offloading methods have different timing requirements and different system benefits. Such benefits may be the saved energy consumption, performance improvement or image qualities. Here, we assume each task τ_i has w_i different versions with respect to τ_i waited to be selected. Suppose the j^{th} version of task τ_i is chosen for the schedulability test, then the demand bound function (DBF) will be presented as $dbf_{i,j}(t)$. Then, as been introduced before, the schedulability test will be given in as below based on the equation (2.1):

$$\forall t > 0, \sum_{dbf_{i,j} \in \Pi} dbf_{i,j}(t) \leq t \tag{3.1}$$

in which Π is the set of selected versions of all independent n tasks. Meanwhile, in order to get the best selection set Π, a parameter $\upsilon_{i,j}$ is also introduced here with respect to task $\tau_{i,j}$. The parameter is presented as a benefit value which will be calculated and accumulated in the algorithm which intends to find the maximum overall benefit. We formally define the problem as follows:

Problem Definition Given a set of tasks τ_1,...,τ_n and each task τ_i has w_i different offloading versions. Each offloading version j from τ_i has a benefit value $v_{i,j}$. How can we select each offloading version for τ_i, $1 \leq i \leq n$ to guarantee all tasks are feasible

according Equation (3.1) and the whole system can achieve the maximal overall system benefits.

Noticed from the prior equation (3.1), it is not efficient and possible to check every time point$t$, especially when the number tasks and versions increases, the exponential growth of the computing time makes it even more impossible to calculate. Here, we use an example to illustrate the difficulties of the problem.

Example1 Given three generalized multiframes Tasks τ_1, τ_2 as follows: Task τ_1 can start execution at any time and generate 3 jobs periodically. J_{11} and J_{13} has different execution time 5 and 3. J_{12} can be offloaded with two sub tasks with execution time 2 and 1, a self-suspension time 4 due to the computation offloading. The interval release time of J_{11}, J_{12} and J_{13} are 10, 12 14. Similarly, task τ_2 can also generate 3 jobs periodically. J_{21} and J_{22} has different execution time 1 and 4. Job J_{23} can be offloaded with a self-suspension time 5 and two sub tasks. The execution time of two sub tasks are 2 and 3. The interval release time of J_{11}, J_{12} and J_{13} are 14, 9 12. The whole execution of the task τ_1 and τ_2 are presented by Figure 3.3.

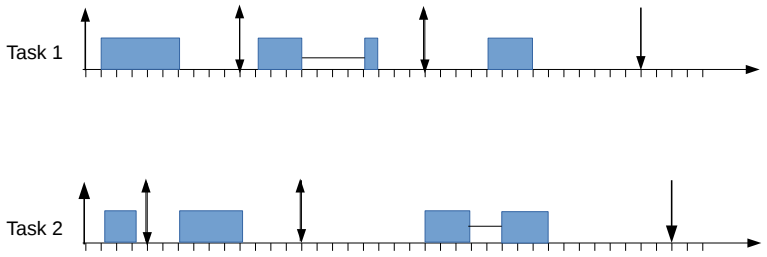


FIGURE 3.3: Time Scheduling for the Three Versions based on EDF

From Figure 3.3, we can see that each task has different execution sequences and the demand bound function of each task can be depicted in Figure 3.4 and Figure 3.5. In order to schedule all these tasks with EDF scheduling algorithm, it is needed to check the schedulability. From the previous analysis, it is impossible to check every time point of all tasks. Hence, it is difficult to solve the offloading decision problem to guarantee that all tasks can be scheduled by the EDF scheduling algorithm.

From this example, we can see that due to the irregular curves of the demand bound function, the offloading decision problem is a non-trivial problem. How to solve such an offloading decision problem is what we need to solve. As been discussed in the previous chapter, schedulability tests for different scheduling algorithms can be really time assuming some even can be *NP*-complete under some specific scenarios such as

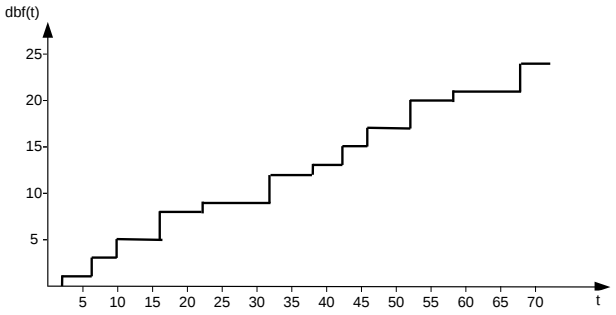


FIGURE 3.4: time demand bound for task1

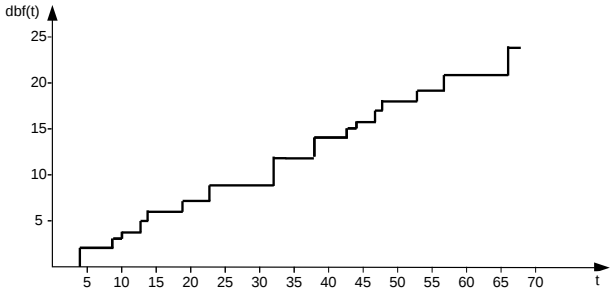


FIGURE 3.5: time demand bound for task2

self-suspension which is the exactly problem will be encountered when one dealing with the realization of offloading mechanism in real time system.

3.3 Approximated Demand Bound Function

Due to the irregular curves of demand bound functions, we propose a new approximated demand bound function which can be adopted here. The approximated demand bound function can simplify complicated representations of the demand bound function. Based on the approximated demand bound function, the complexity of the problem can be largely reduced.

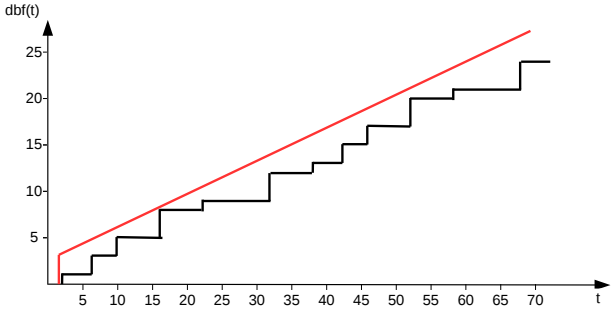


FIGURE 3.6: Example:Approximated Demand Bound Function of Task 1

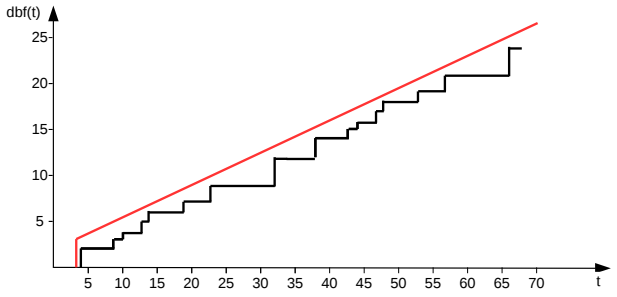


FIGURE 3.7: Example:Approximated Demand Bound Function of Task 2

From the example, one can observe that after some time period or some off-set time point, jumping points of the DBF repeats themselves regularly [29], this can be used for time points checking such that no infinite time checking is necessary. To illustrate the property, equation (3.2) is given below:

$$dbf_{i,j}(t) = dbf_{i,j}(t - \lfloor \frac{t - L_{i,j}}{T_{i,j}} \rfloor) + \lfloor \frac{t - L_{i,j}}{T_{i,j}} \rfloor (dbf_{i,j}(L_{i,j} + T_{i,j}) - dbf_{i.j}(L_{i,j})) \quad (3.2)$$

where $t \geq L_{i,j}$, while $L_{i,j}$ presents the off-set time before the points repeat themselves. Also, given this property, it is reasonable to make the assumption that once these limited points under the schedulability test have been carried out, the restless will not be checked again. It is also true for the ADBF scenario which will be discussed later. Meanwhile, in order to make it more simply to be adopted in the decisions, the discrete jumping points within the demand bound function $dbf_{i,j}$ will be symboled like below:

$$(\eta^1_{i,j}, dbf_{i,j}(\eta^1_{i,j})), ...(\eta^s_{i,j}, dbf_{i,j}(\eta^s_{i,j}))$$

where $\eta^s_{i,j}$ presents the s^{th} discrete jumping time point of the demand function while $dbf_{i,j}(\eta^s_{i,j}))$ presents the demand value. Again take the existed example above, one example result of the Approximated Demand Bound Function can be shown in Fig. 3.6 in which both ADBF and DBF of the task example are completed and compared together. From the figure, it can be seen that the ADBF is a linear approximation of the step curve – DBF like discussed before. The question is how to model such linear function that it can fits the original curve. As been known, a linear function can be characterized with two variables – line slope u and starting point (p, q). In RTOS, the former presents the utilization rate which is critical in the schedulability test. In the next section some algorithms will be presented to model such linear curve and the general idea is to determine the value of the variables.

3.4 HEU Offloading Decision Algorithm

In this section, we propose a HEU offloading decision algorithm as follows:

3.4.1 Design of Approximated Demand Bound Function

To begin with the algorithms used to make the approximation and versions selection, based on the discussion above, the model of ADBF will be set up like below:

$$dbf^*_{i,j}(t) = \begin{cases} 0 & t < P_{i,j} \\ Q_{i,j} - u_{i,j} * P_{i,j} + u_{i,j} * t & t \geq P_{i,j} \end{cases} \tag{3.3}$$

where $dbf^*_{i,j}(t)$ presents the approximated result as been shown above. And it is characterized by three parameters: $u_{i,j}$, $Q_{i,j}$ and $P_{i,j}$. As been discussed, the first one presents the slope of the function which, actually, illustrates the utilization rate if the system. Obviously, the slope of the function should not large than 1 or in case of multiple tasks, the accumulated sum of slope. the point $Q_{i,j}$ presents starting point of the linear function while $P_{i,j}$ presents the time point of $Q_{i,j}$. It is easy to notice that once the three parameters are determined further procedures – schedulability test and versions selection both can be done based on the approximation result.

In order to determine the value of slope $u_{i,j}$, suppose points $Q_{i,j}$ and $P_{i,j}$ are given which will be determined in later section. Under such assumption, the slope value should be set up based on principle that the approximation error between the functions should be as small as possible, one straight forward reflection on the graphic is the linear line – straight line should tight up to the step curve. One example should be like the figure discussed before figure 3.6. Given the principle and assumption, the $u_{i,j}$ value can be determined based theorem below:

Theorem 3.1. *For Approximated Demand Bound Function, given point p, q,* $\forall\ u \in u$

$$u \leq u^* = \max\{\max\{\frac{dbf_{i,j}(\eta_{i,j}) - Q_{i,j}}{\eta_{i,j} - P_{i,j}}\}, \frac{dbf_{i,j}(L_{i,j} + T_{i,j}) - dbf_{i,j}(L_{i,j})}{T_{i,j}}\} \tag{3.4}$$

as been shown above, the slope is determined among those which can be calculated by discrete jumping points. Algorithm 1 shows a detail procedure of determination of

value $u^*_{i,j}$. The input is points $P_{i,j}$, $Q_{i,j}$ and the output is the value of slope. And the algorithm is given based on the equation 3.4 in above theorem.

Algorithm 1 Framework to Build $dbf^*_{i,j}(t)$

$temp\_u = \frac{dbf_{i,j}(L_{i,j}+T_{i,j})-dbf_{i,j}(L_{i,j})}{T_{i,j}}$
$q \leftarrow 0$
while $q < s_{i,j}$ **do**
  if $temp\_u < \frac{dbf_{i,j}(t)-Q_{i,j}}{\eta_{i,j}-P_{i,j}}$ **then**
    $temp\_u \leftarrow \frac{dbf_{i,j}(t)-Q_{i,j}}{\eta_{i,j}-P_{i,j}}$
  end if
  $q \leftarrow q++$
end while
$u_{i,j} = temp\_u$
return $u_{i,j}$

From the algorithm above, it is clear that with given point $Q_{i,j}$,$P_{i,j}$, the value of slope can be yielded with an iteration which tries to find largest value of $u_{i,j}$. Initially, variable $temp_u$ is used to store the temporary value of the result, after comparing with u value yielded with respect with each jumping point, the largest one will be stored and passed to the final value $u_{i,j}$. As been shown above the slope value within the iteration loop is determined by the given two points and each jumping time point and corresponding demand bound function result. This guarantees the fact that the approximated linear demand bound function tights up to the original one which actually makes the corresponding error between them the smallest one.

3.4.2 Schedulability Analysis

After calculation, the parameters of approximated demand bound function are determined and it comes to selection parts. Within the selection parts some algorithms or parameters should be employed as conditions or constrains which models the selection problem. One essential condition is the selected version must be schedulable. In this part, schedulability analysis should be customized such that efficient schedulability tests can be carried on as conditions in the selection procedure.

Given the schedulability tests and especially demand bound function introduced in chapter 2, it is clear that, in principle, the result of approximated demand bound function should never larger than the timing point itself, more straight forward way, the slope should never larger than 1.

Theorem 3.2. *Given a set of tasks τ_1, τ_2,..., τ_n, each task τ_i has a selected version x_i and the corresponding safely approximated demand bound function $dbf^*_{i,j}(t)$ with respect to demand bound function $dbf_{i,j}(t)$. Assume all tasks are sorted in non-decreasing order with respect to the parameter P_{i,x_i}. The task set is schedulable by algorithm EDF if it holds that,*

$$\sum_{i=1}^{j} dbf^*_{i,x_i}(p_{j.x_j}) \leq p_{j,x_j}, j = 1, ..., n \tag{3.5}$$

and

$$\sum_{i=1}^{n} u_{i,x_i} \leq 1 \tag{3.6}$$

As been shown, the first equation (3.5) guarantees the schedulability for starting jumping time points with respect to each task in task set. As to the second equation (3.6), obviously, it simplifies the testing procedures compared with original Demand Bound Function algorithm, only the accumulated slope value should be guaranteed less than one. It also reveals the main idea and application reason of Approximated Demand Bound Function.

3.4.3 HEU-Based Version Selection Algorithm

Followed by section of u determination, this part concentrates on choice of P and Q. For these value determination, it is obvious that one reasonable assumption is the value of P should be less than the first value jumping points $\eta^1_{i,j}$. Specifically, the corresponding time point of these jumping points. Meanwhile the Q value should be the function result value with respect to the first time point of $\eta^1_{i,j}$ – the jumping value of the function. So the question is is it reasonable to set up the assumption like this?

The determination of P, Q value is essential and the procedure is main parameters selection part while it is used to model the function and the starting points are essential for schedulability test and can also effect the error between demand bound function and approximated demand bound function. Firstly, the assumption above guarantees the schedulability property of original demand bound function as the starting points – testing points is less than the first jumping point of the original function. This makes the ADBF is much more reasonable to present the original DBF and a safely approximation.

Moreover, the assumption above actually minimizes the starting offset error which can be seen through the equation (3.3). Also setting the starting point before the first jumping time point makes it more easier to calculate the demand bound function result which makes it more easier for the realization of the algorithm. So it is clear and reasonable that the value of P is less than the x-axis value of the jumping points $\eta^1_{i,j}$ and value of Q is the y-axis value of the point $\eta^1_{i,j}$. Then how to exactly to determine these two value?

Given the assumption above and the theorem 2, it is noticed that the schedulability test will test n different starting points with respect to given the task set τ_1, τ_2,..., τ_n. As been discussed above, the trick is in order to find the possible best $ADBF$ model, it has to be safe and possess the original schedulability property. In other words, the starting point must be less than the first jumping point of demand bound function with respect to the corresponding version. Suppose the starting points is donated as H points, for notational brevity latter, considering the fact that the starting points should be derived from first jumping point, one reasonable and possible propose is choosing the starting points – these H points exactly before jumping points.

Given the H points α_1, α_2,..., α_H, which $0 \leq \alpha_h < \alpha_{h+1}, h = 1, ..., H-1$ the value of points P, Q for given task τ_i version j with corresponding jumping points $\eta^s_{i,j}$ which $s = 1, 2, 3, ...$, can be determined through following procedure:

- We greedily set $P_{i,j}$ to be the maximum α_h among the H points exactly before the first jumping point $\eta^1_{i,j}$ of the corresponding demand bound function $dbf_{i,j}(t)$, i.e., $\alpha_h \leq \eta^1_{i,j}$.
- In order to build a safely approximated demand bound function, one suitable value should be signed to Q. As been discussed above the value should be the y-axis value of first jumping point $\eta^1_{i,j}$. In this way the value $ADBF$ actually should be always less than result of DBF such that it preserves the original schedulability property.
- To simplify the selection procedure, in this section version j should be deleted from versions set of corresponding task τ_i in two scenarios:
  - for those $\alpha_1 > \eta^1_{i,j}$, these make no sense while when all starting points are larger than first jumping point, the $ADBF$ could not guarantee for safely presentation of DBF in time period $[0, \alpha_1]$.

– for those utilization rate yielded from Algorithm 1 is equal or larger than 1. This is obviously not a suitable version selection.

As been seen above, the value determination of Q and P can be achieved by following the procedures. For further illustration of the algorithm, one detail example is produced here:

Example2 Suppose that there are three testing points $H = 3$, $\alpha_1 = 2, \alpha_2 = 3, \alpha_3 = 5$. Again as been like **example1**, here one task set τ_1, τ_2, τ_3 with two different versions are introduced for the demonstration of the $SMP(SelectedMultiplePoints)$ algorithm. Suppose two versions are characterized by their $WCET$ and $deadline$, then for $version1$ they have been presented correspondingly by $E_{1,1} = [3, 1, 2]$ and $D_{1,1} = [5, 4, 3]$. For $version2$ they have been presented as $E_{1,2} = [2, 1, 4]$ and $D_{1,2} = [4, 2, 7]$.

As been shown in equation (3.1), the time points of versions can be calculated easily. For $version1$ the first jumping point in demand bound function is 3 which determines the value of α_2 as the maximum value less than the first jumping time point 3 among H points $\alpha_1, \alpha_2, \alpha_3$, which in turn signs the value of point $P_{1,1} = (3, 0)$. Again, calculated by equation (3.1), it is clear that the first jumping value – also the lowest result of the demand bound function is 2 which value the point $Q_{1,1} = (3, 2)$. Meanwhile after realizing the algorithm 1, the value of parameter $u_{1,1}$ can be yielded easily $u_{1,1} = 0.6$. So the approximated demand bound function $dbf^*_{1,1}(t)$ can be defined as:

$$dbf^*_{1,1}(t) = \begin{cases} 0 & t < 3 \\ 0.6 * t + 0.2 & t \geq 3 \end{cases} \tag{3.7}$$

likewise, for $version2$, the first jumping time point is 2 which signs the value of $P_{1,2} = 2$. The result of the $dbf_{1,2}(t)$ is 1 which determines the value of $Q_{1,2} = 1$. Hence, the result of approximated demand bound function can be defined as:

$$dbf^*_{1,1}(t) = \begin{cases} 0 & t < 1 \\ \frac{7}{13} * t + \frac{6}{13} & t \geq 1 \end{cases} \tag{3.8}$$

Another question is how to determine the value of the H points and what if all the versions proofed to be fail to be schedulable during the parameters selection procedure

as been shown above. While here, the H points are selected randomly at first, then it is possible to find the fact that all the versions seems to be inefficient for schedulability test. This means that the given H points could not used to yield a safety approximation result with $SelectedMultiplePoints$ algorithm and could not be used for calculation. Then the exhaustive search method is used here to change the value and renew the procedure.

Once the parameters are settled, model of $ABDF$ for each version of task set $\tau_1, \tau_2, ..., \tau_i$ will be determined $dbf^*_{i,j}(t)$. All the safety approximated versions are yielded as been shown in prior section **Example2**. Then, the following procedure is to select the best combination among the result versions. In this section, the problem will be modeled first, then one algorithm named HEU is introduced to address the model, finally one best solution of version will be yielded for offloading mechanism realization.

First, as been discussed, the selection problem needs to be modeled mathematically. The usual procedure for such best version finding problem is try to define a benefit function with relative variables and try to find the optimal value of the parameters for minimization or maximization the benefit function. In the project, the benefit function can be defined as the overall benefit – the accumulated benefit of each task. Let $x_{i,j}$ be a versions selection decision parameters which specifies the selection situation with respect to the task τ_i version j. It is valued as $x_{i,j} \in \{0, 1\}$. In other words, when $x_{i,j} = 1$, then the task $\tau_{i,j}$ is selected. As been shown in prior section, for each task in task set, it is possible that each task possess different number of versions needed for selection. Hence, for notational brevity, the feasible approximated demand bound function versions yielded by algorithms introduced in last section are indexed as $1, 2, ..., w'_i$, which, obvious, w'_i may differ from w_i value introduced before.

Another parameter needed to be introduced for problem modeling is benefit variable υ, which presents the benefit of offloading the selecting task version of j. This parameter can be valued either through system resource saving – time saving due to the offloading mechanism in the project, or quality comparison between results of task carried on local computer and those on remote server computer. Obviously, the server possesses more resource and produces higher quality result. Due to this benefit parameter definition, it can be noticed that the more tasks are uploaded to the server, the more benefit can

be yielded. Schedulability, however, may be an issue when all tasks select the version possesses the best benefit. This is also the significance of offloading decision problem.

Once the benefit parameter υ, the benefit function can be defined as follow:

$$f(x_{i,j}) = \sum_{i=1}^{n} \sum_{j=1}^{w_i'} x_{i,j} \upsilon_{i,j} \tag{3.9}$$

as been shown above, the versions selection problem is modeled through this benefit function. Obviously, to find the best solution of task versions is to maximize the function result with respect to parameters – task selection decision parameter $x_{i,j}$. Once the value of $x_{i,j}$ is determined for each task, it is clear that which version should be selected for the entire task set benefit function.

For maximum benefit function, constrains must be considered, otherwise solution may be optimal but unreasonable or unrealizable. In the project, such constrains lies in the schedulability and accumulated value of variable $x_{i,j}$. For schedulability, as been introduced in theorem 3.2. The equation (3.5) and equation (3.6) can be employed to define the constrains here, specifically:

$$\sum_{i=1}^{n} \sum_{j=1}^{w_i'} x_{i,j} dbf_{i,j}^{*}(\alpha_h) \leq \alpha_h, \forall h = 1, ..., H, \tag{3.10}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{w_i'} x_{i,j} u_{i,j} \leq 1 \tag{3.11}$$

equation (3.10) describes the schedulability test for the given H points, which has been discussed before. The second equation (3.11) presents the accumulated slope – the overall utilization rate of the system. As been illustrated in equation (3.6), it must be less than 1.

The last constrain is simple as for variable $x_{i,j}$, it is obvious and reasonable that one task should only select one version. For the mathematical expression, it should be described

as below:

$$\sum_{j=1}^{w_i'} x_{i,j} = 1 \qquad x_{i,j} \in \{0,1\} \tag{3.12}$$

Once the model is set up, algorithms should be implemented to address the problem. To find out solution of the problem, a further explanation is given here. As been discussed before, the aim of the algorithm is trying to find the best accumulated benefit. Meanwhile, the overall utilization constrain should not be violated. When the number of tasks and versions increased, the calculation should increase explosively. This is not suitable for programming and realization in embedded system. This kind of problem can be named as *MMKP (Multiple dimensional Multiple choice Knapsack Problem)*, obviously, it is an $NP-hard$ problem. The computational time may grow exponentially with the growth of task set scale.

One simple example is to fulfill a container with multiple small cubes. As been shown below, it is possible that for each cube there are several options which possess different properties, in this scenario, different length, heigh and width. Different options produce different rewards. The *MMKP* problem is to find the highest value of reward with out violating any constrains, in this scenario, the total length, heigh and width must be less than those properties of container itself.

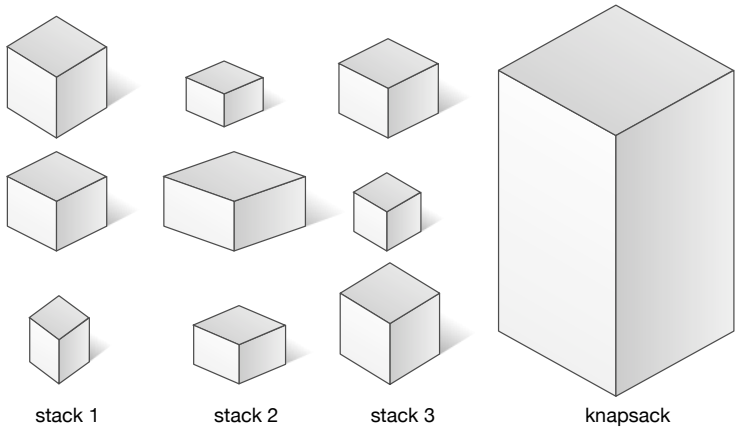


FIGURE 3.8: One Example for *MMKP*

One solution for such problem is to adopt a heuristic HEU algorithm to address the problem. The algorithm yields aa fast and near-optimal solution of the *MMKP*. As the

problem concerns the resource allocation within RTOS. Some variables can be introduced first for problem modeling. Suppose within one task set $\tau_1, \tau_2, ..., \tau_n$, each task possesses optional versions j, which $j = 1, ..., w_i'$. Suppose $\mathbf{C}$ and $\mathbf{R}$ present the current system resource vector and total system resource limit vector respectively, which means that perhaps there are more than one system resource constrains, not only timing constrain. In order to present the system resource consumed by version j of task τ_i, symbol $\mathbf{r}_{i,j}$ is introduced here. Noticed that variable $\mathbf{r}_{i,j}$ is a vector due to the multiple system resource while symbol $r_{i,j}$ only represents the aggregate resource consumption proposed by Toyoda. The aggregate resource consumption is introduced here for a quantification of resource changing due to the alternative selection of the versions. As for versions selection, as been introduced before, $v_{i,j}$ presents the value for each task version while $\rho[i]$ reflects the selection decision with signed value $\rho[i] = 1, 2, ..., w_i$, with respect to task τ_i. Meanwhile, in order to make an iteration, parameters $\triangle r(\rho, i, j)$ and $\triangle p(\rho, i, j)$ present the resource changing value and value gain per unit of extra resource due to the updating of version selection.

To summarize the forementioned legend and further illustrate the algorithm, the main procedure will be described and a pseudo algorithm will be given later. Now the main procedure is presented as below:

- The algorithm starts with picking up the versions with smallest value $v_{i,j}$ and replace the selection iteratively with those possess higher value as long as the selection task set satisfy all forementioned constrains –feasible for scheduling under EDF in RTOS.
- With conception of aggregate resource consumption mentioned by Toyoko, it is possible to present the system overall resource in form of scalar instead of vector comparison when dealing with multiple resource constrains.
- For updating principle, the main idea is replacing current selection with those which maximizes the saving in aggregate resource. If no such item is found, then one with higher value of gain per unit of aggregate resource is chosen.

For specifically procedure and further illustration, the pseudo algorithm is given in algorithm 2. As been shown, the input of the data is properties of task versions and the output is the best selected combination.

To summarize the data processing procedure mentioned in this section, and make a preparation for the next section – UI realization, a processing pipeline is introduced here. First, in order to schedule the task set with self-suspension cased by offloading mechanism, a Approximated Demand Bound Function is derived from the input variables which characterizing the independent task set – $WCET$, deadline and period. Once the $ADBF$s are yielded by algorithm 1. As been shown before, the demand bound function is then characterized by three parameters which should safely guarantee the original schedulability properties. Then all these versions of tasks are valued with corresponding penalties which present the *value* of each task from perspective of users. This leads to the version selection which can be seen as a $MMKP$ problem. With given parameter – slope of $ADBF$ $u_{i,j}$and each version value $v_{i,j}$, algorithm HEU is proposed to select the best solution of the problem. In the end, with optional selected and schedulability proofed versions, the task set should be safely carried on in real time embedded system under EDF algorithm.

To make the forementioned algorithms more manipulatable and generalizable, one demo will be presented in next section to realize those data processing procedures. More details and main procedures of realization will be discussed then.

3.5 Priority-Based HEU Offloading Decision Algorithm

As been shown above, to approximate demand bound function, parameters starting point P, jumping point demand Q and approximated curve slop u are essential. Starting points P will produce an distance error which effect final judgement of schedulability. So in this section one classification algorithm is proposed to reduce such distance error.

The best solution, obviously, is that we can determine each approximating curve function with exactly same value of the first jumping point. One problem, however, may arise that such solution may lead to n dimensional $MMKP$ problem which will not be seen as a good solution. Suppose n tasks needed to be scheduled, then n starting points are determined through such way. In order to solve the scheduling problem, algorithms like

Algorithm 2 *HEU* Algorithm

/* A heuristic algorithm for MMKP */
/* Legend:
n: task set number w_i' version number for task τ_i i, j: task i version j
m: system resource number $\mathbf{C}$: current accumulated resource v: value
R: total resource limit $\rho[i]$: solution vector $\mathbf{r}_{i,j}$: resource for each version
$\triangle p$: value gain per unit of extra resource $\triangle r$ aggregate resource saving */
procedure HEU:

1: **for** $i = 1, ..., n$ **do**
2: $\rho[i] = 1$;
3: **end for**
4: $C = \sum_{i=1}^{n} \mathbf{r}[i][\rho[i]]$
5: **while** 1 **do**
6: $\triangle p_{max} = 0 \triangle r_{max} = 0$
7: **for** $i = 1, ..., n; j = \rho[i] + 1, ..., l_i$ **do**
8: **if** $\exists k : k = 1, ..., m, C[k] - r[i][\rho[k]] + r[i][j][k] > R[k]$ **then**
9: *continue*;
10: **end if**
11: $\triangle r = \frac{(r[i][\rho[i]] - r[i][j]) \cdot C}{|C|}$
12: **if** $\triangle r_max < \triangle r$ **then**
13: $\triangle r_max = \triangle r, i' = i, j' = j$;
14: **end if**
15: **if** $\triangle r_max < 0$ **then**
16: $\triangle \rho = \frac{(r[i][\rho[i]] - r[i][j]) \cdot C}{\triangle r}$
17: **if** $\triangle \rho_max < \triangle \rho$ **then**
18: $\triangle \rho_max = \triangle \rho, i' = i, j' = j$;
19: **end if**
20: **end if**
21: **if** $\triangle \rho_max <= 0 and \triangle p_max <= 0$ **then**
22: $return \rho$
23: **end if**
24: $C = C - r[i'][\rho[i']] + r[i'][j']$
25: $\rho[i'] = j'$;
26: **end for**
27: **end while**

HEU are carried on whose constrains number will be signed by number of P. So simple adopting value of jumping points seems fail to address the problem.

Alternative solution like determining all starting points with simple one or limited values alleviates the calculation pressure as it reduces multiple constrains into only one or limited constrains. The disadvantages, however, is quite obvious that such method may increase error between original BDF and approximated DBF.

In order to determine value of P, meanwhile, avoiding increasing approximated error, one classification procedure is proposed here. Basically, in order to sign the value of each

starting point, the first jumping points of each demand bound function are classified into m groups and the P value of each DBF will be valued with the smallest value in each group. This value m can be set up randomly. Obviously, increasing number of groups may reduce the error while it will increase the constrain dimension at the same time.

It can be noticed that such method is basically adopting classification algorithms which are widely used in unsupervised machine learning problem. One example is K-Means algorithm. Here, to offer a customized algorithm with respect to real time tasks scheduling, algorithm 3 is introduced here.

Algorithm 3 classification Algorithm

```
/* A classification algorithm for determination of P point */
/* Legend:
     n: demand bound function number
     m: signed classification number
     point_(2): vector stores first jumping points
     label: label of points position aggregate resource saving */
for i = 1, ..., n − 1 do
  for j = 1, ..., n − i − 1 do
    if point[j] > point[j + 1] then
      point[j] ⟷ point[j + 1]
      label[j] ⟷ label[j + 1]
    end if
  end for
end for
for i = 1, ..., n − 1 do
  sub[i] = point[i + 1] − point[i]
end for
threshold = sub[m − 1]
for i = 1, ..., n − 1 do
  if sub[i] < thrshold then
    point_2[i + 1] = point_2[i]
  end if
end for
for i = 1, ..., n do
  Parameter_P[sub[i]] = point_2[i]
end for
return Parameter_P
```

As can be seen above, in the first part, all first jumping points have been sorted in increasing order with Bubble Sort algorithm. Then subtraction between adjacent elements is carried out and sorted in decreasing order. Based on selected classification number m, the mth subtraction value is selected and seen as threshold. According to this value

and subtraction results, elements can be signed with the lowest value in each group. In the end the result of parameter can be returned.

It can be noticed that the algorithm adopted Bubble Sort method which is the most time consuming part. So considering the worst case, the complexity of the algorithm 3 is $O(n_2)$.

The advantage of HEU algorithm is it tries to find out optimal solution in a greedy way, which trends to solve the MMKP problem in shorter time compared with non-HEU algorithm. One disadvantage is the searching direction or iteration policy seems deficient. This is due to the fact that the knowledge of candidates can not be known. Any favor in any searching direction may or may not yield a quicker answer to the question.

Considering about the fact that in RTOS, applications or tasks can be signed with different priorities which is quite common under resource limited circumstances. Even for detection tasks, for example, different detection objects can be sighed with different priorities according to the situation. In this way, the approximated demand bound function can be also signed with such priorities. This can help to improve HEU algorithm.

With priorities signed with each task, the searching direction or as mentioned iteration policy should favor tasks with higher priorities. In other words, resources assignment will be considered first with those of higher priorities tasks. Algorithm 4 illustrates the basic processing procedure of such method. Suppose the tasks are scheduled in priority decreasing order with value from n to 1, while the versions are scheduled in penalty increasing order with value from 1 to w, in which w presents number of version and n indicates task number, then the result can be yielded through following algorithm.

In the first part, a greedy result is chosen as initialization value which is the best result only considering the penalty. Then a *while* loop is carried on until possible result is found. This is realized through *break* option. As can be seen the second part – loop body can be separated into two sub parts: **(1) constrains checking; (2) break the loop or continue iterating result**. As noticed, the principle behind this is as same as HEU algorithm, iterating result until optimal solution appears. The difference is the iteration policy in this algorithm guarantees that the iteration candidates are of penalty

Algorithm 4 HEU Algorithm with priority

```
/* Priority based HEU algorithm */
/* Legend:
     n: tasks number
     w: versions number
     flag_schd: flag of schedulability
     acc_point: detection points
     acc_value: accumulated value of detection points
     result: vector stores final optimal result
     label: label of points position aggregate resource saving */
1: for i = 1, ..., n do
2:    result[i] =versions with highest penalty
3: end for
4: while 1 do
5:    if Parameter_u[result[i]] >= 1 then
6:       flag_schd = 1; break
7:    end if
8:    acc_point = Parameter_P[result[i]]
9:    acc_value = 0
10:   for l = 1, ..., n do
11:      acc_value+ =point value
12:   end for
13:   if acc_point < acc_value then
14:      flag_schd = 1; break
15:   end if
16:   if flag_schd = 0 then
17:      break
18:   end if
19:   if flag_schd = 1 then
20:      flag_schd = 0
21:      for i = n, ..., 1 do
22:         if result[i]%w=0 then
23:            continue
24:         end if
25:         result[i] = result[i] − 1; break
26:      end for
27:   end if
28: end while
29: return result
```

& priority decreasing order. In other words, if the possible solution is found, then it is the optimal one.

Due to the fact that this algorithm is also HEU based algorithm although the iteration policy, as forementioned, determines the researching direction and speeds up the iterations. In the worst case, the algorithm complexity will be $O(wn)$.

Chapter 4

Case Study Virus Detection

As been shown in the prior chapter, the aim of the thesis is to realize an offloading mechanism in project – "Virus Detection". In this chapter, more details about the project will shown and a demo will be presented to illustrate the mechanism. As for the offloading decision and parameters determination, they will also be introduced in latter section.

4.1 Techniques about Virus Detection

Virus, based on definition of Merriam-Webster, is an extremely small living thing that causes a disease and that spreads from one person or animal to another. Few, but, horrible instincts such as smallpox, pandemic, AIDS, or even recently Ebola virus, have shown that it can lead to countless losses and leave nothing but ruins and panic, a soundless massacre. Meanwhile, people's pursuer towards simple and efficient approaches of virus detection never stops, which not only protect us form infection but also may help us find out "Patient One" with certain algorithms. The intention of this project is to make the detection feasible on a mobile embedded device which should make it more portable and easier. The project aims at detection of biological viruses using portable biosensor with fuzzy-enhanced algorithm.

4.1.1 Background of Virus Detection

Rising numbers of global virus epidemics increase the demand for infection control. It is desirable to deliver diagnoses on-site and as quickly as possible, in order to prevent further spread of virus-transmitted diseases. This calls for virus detection devices which are fast and portable. Such devices can be used e.g. at airports, to answer the question whether or not passengers might propagate contagious diseases from high risk regions. A novel method providing the basis for a portable and real-time capable virus detection device is the so-called PAMONO technique (Plasmon Assisted Microscopy of Nano-Size Objects). It enables selective detection of different types of nano-objects, including but not limited to viruses. In order to be detected, the nano-objects must be immobilized on the surface of the PAMONO biosensor. In the case of viruses, this is achieved by preparing the sensor surface with antibodies. Using different antibodies enables distinction of multiple strains of viruses. Conversely, it it possible to determine which antibodies are capable of attaching to a certain kind of virus. These capabilities are combined in an inexpensive device, consisting of the biosensor and of a laptop computer for data analysis.

4.1.2 Virus Detection based on PAMANO Sensors

In 2010 a novel device named PAMONO (Plasmon Assisted Microscopy of Nano-Size Objects) was proposed (Weichert et al., 2010; Zybin and et al., 2010[30]) to perform a basis portable real-time capable virus detection. Instead of hours labor in the lab, the detection procedure can be achieved by general medical practitioner holding mobile devices anywhere anytime. It enables detection of different nano-objects, not only virus, but also can be applied into field of environment monitor, automobile exhaust, for instance.

PAMONO sensor, short for Plasmon Assisted Microscopy of Nano-Size Objects sensor, detects nano-objects based on the differential reflections of detected objects compared with the surroundings. The general procedure behind it is utilizing a gold layer, which will be used to generate a surface plasmon resonance effect, to immobilize the detected virus with corresponding antibodies, then the fixed virus and antibody will be illuminated by a diode behind the golden layer. Then, according to the surface plasmon

resonance effect, the "surface plasmon resonance properties will change within a micrometer scale area around that nanometer scale attachment". This results in an increasing reflectance of the sensor surface in that micrometer scale area. More light from the diode will be reflected at the attachment site, which can be detected using optical microscopy"[3].

At the same time a highly sensitive 12-Bit CCD camera chip will be used to record the reflection of the effect through a series of images with a lateral resolution of 1024 x 256 pixels at a temporal resolution of 30 frames per second. Figure 4.1 first part illustrates the main structure of the device PANOMO. If the images were received in time, then they will be seen as input variables and put into algorithms that can be applied to make a detection and classifications based on given image features corresponding to different demands.

Figure 4.1 illustrates the main idea of the detection procedure. It is clear that the general procedure can be roughly divided into two parts – recording images through PANOMO sensor, analyzing data with received images. Due to PANOMO sensor's hight analytic sensitivity, it is feasible to get enough features to make further decisions associated with algorithms such as fuzzy-enhance detection technology. After that the result will be yielded immediately to tell whether the images contain a virus or not. So in conclusion, the basic procedure, a PMONO sensor based virus detection technology, can be listed as follows:

- immobilization of detected virus through corresponding antibody
- record images of reflections from diodes
- data analysis

As can be seen, the detection procedure also can be regarded as a real time operating system while the procedure can be seen as an online processing procedure. Although powerful computing components can be utilized to guarantee the result due to the adoption of offloading mechanism, missing deadline, somehow, can still occur given consideration of problems such as self-suspension. RTOS computational offloading issue will be discussed in detail and fair solution will be given.

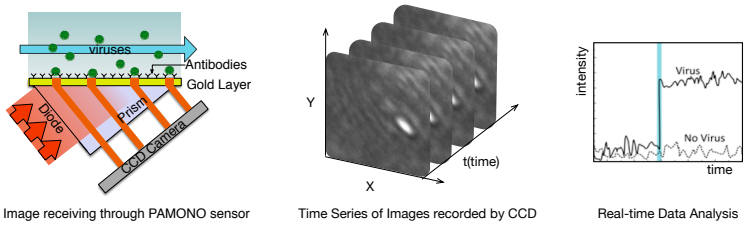


FIGURE 4.1: PAMONO biosensor (left), the recorded data (center) and the concept of virus detection (right)[3]

4.1.3 Data Analysis about Virus

After receiving images, a four-step fuzzy-enhanced algorithm will be adopted in order to yield a final solution. Figure 4.2 reveals the detail of the four-step processing algorithm.

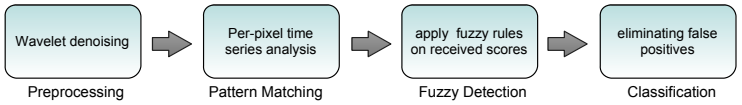


FIGURE 4.2: processing pipeline for biosensor data analysis[3]

As can be seen, the procedure is divided into four steps: preprocessing, pattern matching, fuzzy detection and classification. For any single pixel on the frame of the time series recorded images, the detection algorithms will be carried out in both temporal and spatial direction. It can be seen as a detection pipeline, which processes every incoming image and detects every single pixel on it. Specifically, in the first processing pipeline, different denoising techniques will be applied here aiming at noise removing. Random impulse noise removal, for example, uses fuzzy logic technic to make the noise removal more efficient. After that several other image processing algorithms will also be used to perform an image enhancing procedure, which mainly make a preparation for the next step, pattern matching.

In the pattern matching procedure, a 3D volume of matching scores will be yielded with respect to each pixel after comparing time-series frames. In general case, this step will generate a score with respect to each pixel. It is designed to value the existence possibility of the virus. The common approach is using a hard threshold with respect to the selected features. The problem is sometime this may lead to missing detection or over detection due to failure determination of the threshold. In the project, however, this will be released by adopting a fuzzy logic enhancement method, which yields a more

robust and exact detection result of telling whether the pixels contain a virus candidates or not.

Moreover, an auxiliary procedure will also be added here to make an additional verification to eliminate the false candidates. This process can also be seen as a feature based classification. In the project, some shape related features will be used to make the classification. Specifically, the contiguous areas of each candidate pixel will be aggregated to polygons, which will be used as features in a random forest classifier to distinguish actual viruses from artifact detections. A clear illustration will be seen through the Figure1.4. From the figure, inputs and outputs of each step will be figured out and compared.

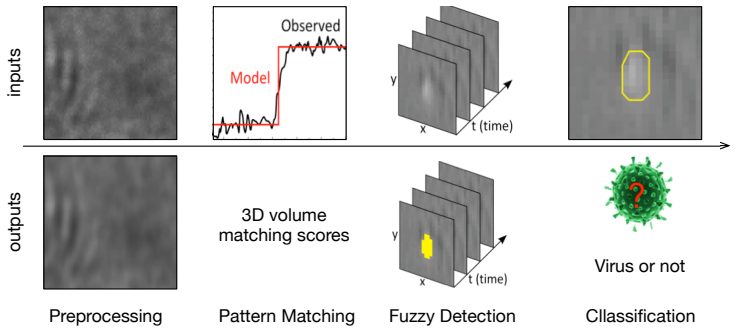


FIGURE 4.3: inputs and outputs of processing algorithms.[3]

More detail about the algorithms could be discussed in latter chapters, where methods will be proposed and analyzed to realize offloading decisions.

4.2 Portable Virus Detection Device

In the project, the detection part and algorithms will be accomplished and simulated first with computer software *Qt* which mainly performs functions of the mobile interface and parameters configurations. Then the program should be carried out on application board with a live CCD camera, with which the system can be then considered as a real time system. After setting up of the environment, parameters selection, server and client setting up, the offloading should be carried out. Firstly, basic communication between client and server should be set up based on socket network programming, which will be discussed and presented in later chapter. Moreover, studies about offloading decisions should be carried on aiming at finding solutions with respect to different scenarios such as offloading with only one task or offloading with multiple tasks, which basically, will be

the general case in RTOS. Meanwhile, offloading decision problem should be considered here to find the optimal solution of which part of algorithms should be uploaded. Given proposed solution, a simulation will be applied to illustrate the idea and schedulability should be discussed and guaranteed. With no doubt, considering about problem of self-suspension, offloading mechanism will cause a scheduling problem. Then, specific algorithms will be presented to address the problem proposed above along with a interface designed with Qt to illustrate the idea. Latter, an interface will be designed to show the result of the offloading mechanism decision which will be generalized for problems such as versions selection problem. In the end results will be yielded and analyzed, further discussion will be given.

4.2.1 Software Design

Here, we present the software architecture of the portable virus detection device. In this architecture, we have the hardware layer, operating system layer and application layer. In the hardware layer, the sensing parts is composed of PAMANO sensors and CCD cameras. The PAMANO sensors and CCD cameras are connected with an embedded system. The embedded system has some powerful CPUs and GPUs, which can also process some virus detection algorithm locally. There are also some communication modules to enrich the functionality of embedded systems, such as WIFI, Optical Fiber and etc. In the operating system layer, we can utilise some general-purpose operating systems, such as ubuntu. Moreover, in order to guarantee the real-time response time, there are also some real-time operating systems which can also be used. In the application layer, the embedded system can finish some image acquisition, de-noising and etc.

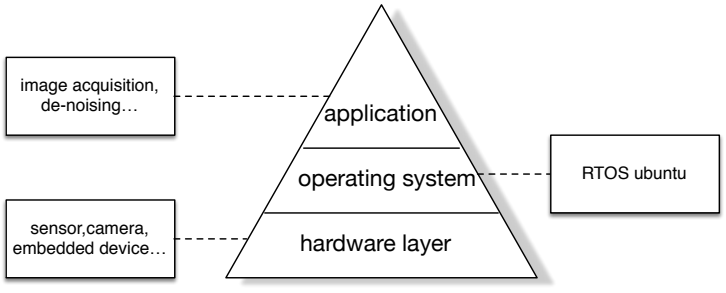


FIGURE 4.4: software design

For example, if we only need to detect limited virus types, the embedded system can finish the whole detection. However, when the embedded system needs to process several

different virus types at the same time, the local embedded system with limited computing resources is difficult to finish all computations. At that time, a remote server can be used to increase the computing capacity and storage.

From the application of algorithms, in the virus detection algorithm, according to different parameter configurations of our system, each application can also have different ways to offload computations.

4.2.2 Related Tools

In this part, we introduce some related works which are related to our system implementation as follows:

First, our implementation is based on a cross platform application framework QT. The QT framework has several functions which makes the virus detection algorithm much more easier to develop with graphical user interfaces (GUIs). One of the important properties of QT framework is that it supports standard C/C++ library. Gstreamer and OpenCV libraries can also link to the QT application. Moreover, QT framework has the property which can process many multimedia applications. A novel concept introduced in Qt — "signal and slot" makes image acquisition much more easier. communication between objects more easier.

Gstreamer is another widely used multimedia libraries, which are based on a pipeline-based architecture. In Gstreamer, a component named element is set up first with source and sink which define the data flow. Then different elements with different functions accordingly, will be connected together to form a pipeline which can be seen a top-level collection of elements. Obviously, with different functional elements, multimedia can be processed step by step through data flow. In the thesis, one pipeline is set up to handle a data processing procedure which starts from input data buffer and ends with output final decision.

OpenCL is a programming framework which is based on heterogeneous platforms, which can be used in single processor, multiple processors and heterogeneous processors. In reality, due to some complicated parallel computations, OpenCL can take the advantage of GPUs to accelerate the application execution, which can solve the resource limited problem of embedded systems.

Besides framework mentioned before, another library named OpenCV is also used in the project to process image and video. OpenCV (Open Source Computer Vision Library) is also a cross platform computer vision library which is widely used in image processing, computer vision and model recognition. Specifically speaking, the library offers hundreds of algorithms which not only have c/cpp interface but also those for programming languages such as Python and Java. These algorithms make it more easier to process multimedia, images for example. Moreover, the most exiting thing about it is that it can really realize a parallel processing mechanism which can significantly speed up the calculations combined with OpenCL. This makes it more suitable for real time computation tasks in which scenario timing requirements should be paid more attention. Through acceleration of hardware and software architecture, tasks can be guaranteed to be accomplished within given constrains. That is the essential idea of real time operating system encountering with complicated computations. In the demo of latter section, some image processing algorithms from OpenCV are adopted here to do a preprocessing of images. More detail will be discussed later in next section.

4.2.3 Software Implementation

In the client side, video images will be collected from a CCD camera which has been introduced along with the PAMONO device in the first chapter. Then the first image processing pipeline — preprocessing pipeline is carried on to make some preprocessing procedures which will be helpful for latter processing. In this part, some universal preprocessing algorithms are adopted to make the detection more efficient and feasible. In this part the video data received form camera will be transferred and converted from *Gstreamer* buffer into image processing data buffer, which can offer the inputting images for latter processing. This will be realized with *Gstreamer* architecture and corresponding library. Also the algorithms are scheduled in form of "plugin" in the pipeline mentioned before in section *Gstreamer*. The advantage of this architecture is it will be more flexible to schedule the algorithms while as "plugin", they can be easily added, removed or changed during the image processing procedures. Also all states of data can be seen through calling functions of *Gstreamer*. The image data flow in through "source" port of one element and be passed to another through "sink" port. Fig. 4.5 below will illustrate the structure specifically.

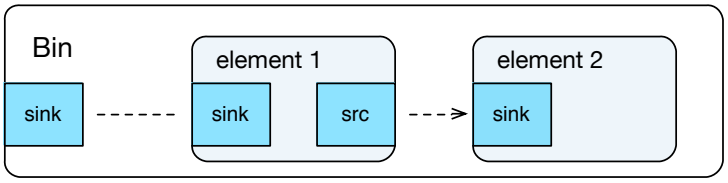


FIGURE 4.5: structure of "element" in *Gstreamer*

From the figure, it can be seen that the source data flow into processing element from original source data, and then are passed to the source ports of elements to be processed by algorithms, then send to following elements though port sink. Likewise, all algorithms are presented as "plugin" and image data pass through all elements which can be seen as one big "pipeline". The data follow pass by and the final detection – existence of virus can be yielded and returned.

To accelerate the processing procedure, *OpenCL* library is linked into *Gstreamer*. To realize that, a "gst-plugins-cl" is applied. Also, some common image processing algorithms can also be called through library *OpenCV*. In the end, to integrate the whole project, software *Qt* is applied to make it an application. Fig. 4.6 below reveals the whole software structure of virus detection procedures

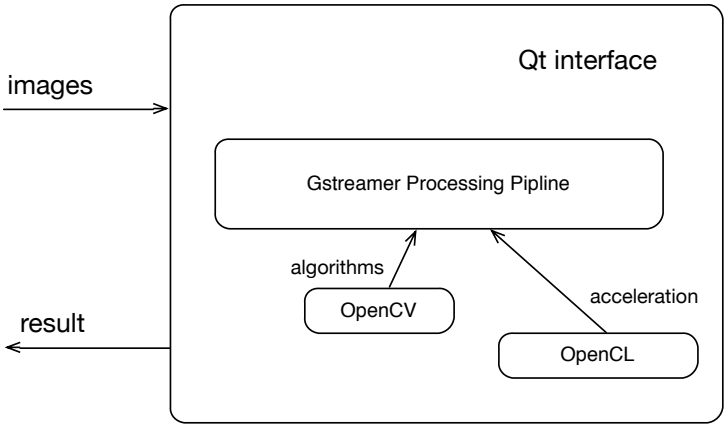


FIGURE 4.6: software structure

From the image it can be seen how algorithms are scheduled together and how these softwares are associated with each other.

Once semi-processed images are uploaded into remote server, the rest processing procedure will carry on, the main processing procedures have already been introduced before. The details can be found in paper [3].

4.2.4 Experiment

In reality, to perform such detection functions, client and server platforms are needed. Here, the client configuration is 2.9 GHz Intel Core i7 with Intel HD Graphics 4000 1024 MB. While for server, there are eight Intel Xeon ES-2609 v2 2.50Hz with four AMD Radeon HD 7900 Series.

In the client side, QT software framework software is adopted. Some image processing algorithms, such as , are employed based on OpenCV and QT.

At first, given the function of the program – periodically receiving data from camera, processing the yielded images and realizing the offloading.

It can be noticed that in the main program, functions should be called periodically to read the image data and then send them to the processing procedures. During the procedures, image denoising or some other image processing algorithms are adopted. Finally, the semi-processed images are uploaded.

Back to Qt, such structure is realized through adopting *signal* system. *Signal* system can be seen as a "callback" function which can be triggered with multiple actions or other signals. Basically, it can be seen as an event or time triggered based function. The principle idea is to connect a $SIGNAL$ sent trough a QT widget with a $SLOT$ generated by functions. Once the signal is sent, the function can be executed then. One simple example can be one Qt widget $PushButton$, once the action "$Push$" of the widget is triggered, the corresponding signal will be sent and the "signal function" will be executed then. In the case of reading image data from camera, the reading function $read()$ is offered by video processing class $VideoCapture$ in $OpenCV$ which deals with extracting image frames from camera, video or web. The output of the read function is stored in form of matrix. This process will be carried on periodically. As been mentioned before, in order to trigger such image reading process, one simple solution is to adopt $Timer$ widget. The class $QTimer$ offers time function which enables one to read and control timing parameters. Specifically, with the $Open(int)$ function in the class, the timer will be triggered with respect to the setting parameter. Then the sending signal can be connected with $SLOT$ of image processing functions. Once the signal is sent, the functions will be carried on. In this way, the image reading from camera can be processed periodically.

In the image processing functions, some basic algorithms are realized. First, considering about **image cutting** procedure. When dealing with multiple detection tasks, resource limitation should be the main problem and image processing procedures should be designed to alleviate system computational pressure. One possible method is image cutting procedure. The object of such procedure lays in two considerations. Resource saving is the first reason. The second one is the image quality of virus candidates decreases due to the existence of angle between light beam of diode and CCD camera. This may lead to bad result of detection on the edge area of the received images. So instead of processing whole images, the detection results mainly depend on virus candidates in the central area of images. Fig. 4.7 illustrates the receiving images from CCD camera.

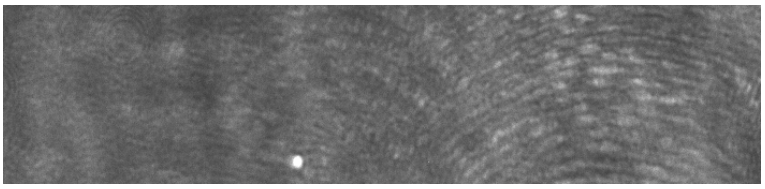


FIGURE 4.7: real received image from CCD camera(1080,145)

As can be seen, the size of received images form CCD camera is (1080,145). To realize the image cutting procedure, the size of image should be determined first. To differ different detection execution time, we suppose for those viruses with larger diameter, less edge area will be cut off and vice versa. The cutting can be done through simply valuing the Mat format variable in $OpenCV$. With different size of images, different detection tasks may possess alternative execution time.

The second step of procedure is **image denoising**. While for this specific virus detection tasks, the denoising procedure can be done with following procedures. test Considering about the fact that the temporal solution of the CCD camera is 30 frames per second, the image slight changes over time. Then, to accelerate the processing procedure, an image buffer is set up. The buffer is illustrates through the following figure.

As can be seen, denoising procedures can be divided into three sub-steps according to the data collection. The first and last m images and n images in the middle. So basically, in order to save system resources and realize the denoising function, the middle n images are skipped over while the first and last m images can be combined through averaging images to yield two new processed images image Im_a and image Im_b respectively. Then these two images can generate the final denoised images through the following formula:

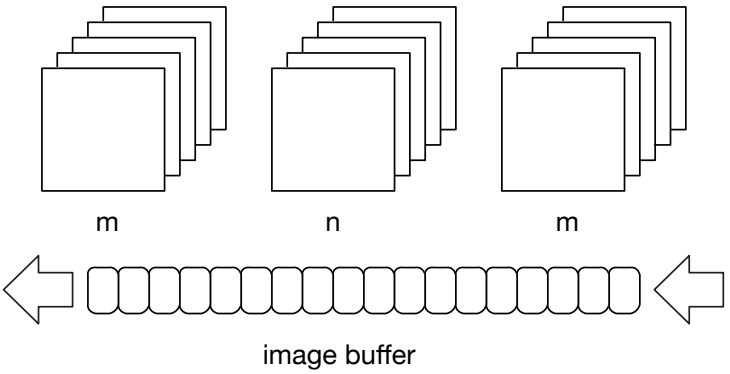


FIGURE 4.8: coming image buffer

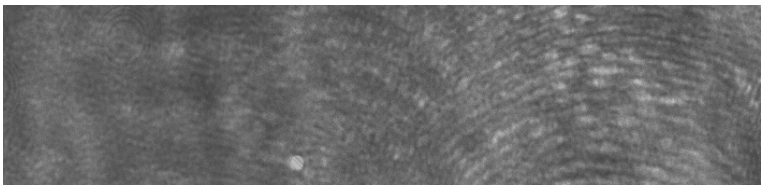
$$D(i,j) = \frac{Im_b(i,j)}{Im_a(i,j)} \tag{4.1}$$

in which D stands for the demoised image while i, j present the position of each pixel. In such way, for each coming image form camera, one denosed new image can be yielded with $m + m$ images through such buffer. Normally, we set up value with $m = 10$, $n = 8$. Fig. 4.9 illustrates the corresponding results. As the images slight change, the candidates are difficult to tell, to illustrate these points, in the following image, the background is set to white and the image is zoomed in.

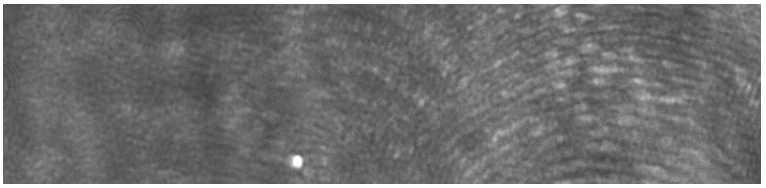
As seen, image a and image b are the results of averaging images of first and last m images. The last image is the final result of denoised image. During such processing, however, another method can be adopted to accelerate the processing procedure – **image skipping**.

As mentioned before, the processed images will be yielded and uploaded to remote server after the image buffer is full. In other words, with one coming image, one result image is yielded even through it covers the information of imaged in the buffer. The trick is we can set up the processing procedure to skip l images each time, that means for each incoming l images, we will produce one denoised image. In such way the system can be accelerate and system resource can be remained.

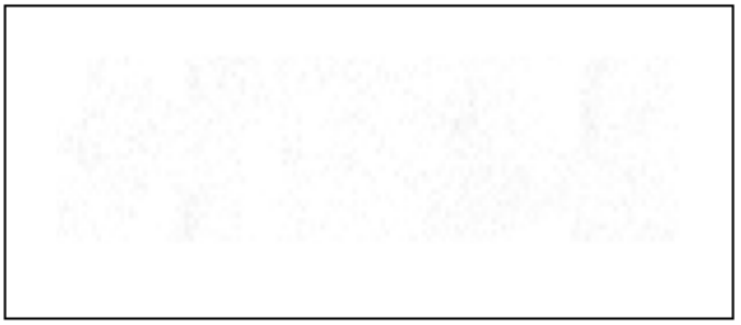
In the end the processed images are uploaded to remote server with socket programming. Further processing procedures can be carried on with more powerful computational components.



(a) average image Im_a



(b) average image Im_b



(c) denoising image D

FIGURE 4.9: result of image denoising wit forementioned buffer

With uploaded processed images, further procedures can be carried on on the server. As been discussed before, the application number can be more than one which means the scheduling algorithm should handle such scenario. In the next chapter, parameters of determined model will be determined and solution of task scheduling problem will be given.

Chapter 5

Experiments & Results

In the prior chapter, models of offloading mechanism have been discussed and one demo has been presented. Simple application with offloading mechanism can be easily done with socket programming and disadvantages have also been illustrated. Meanwhile, for more complicated model – multiple applications with offloading, more sophisticated solution should be introduced. This chapter concentrates on multiple applications model and one final optimal solution will be shown in the end for tasks scheduling issue.

5.1 Model

As been shown, it is possible and efficient to execute more than one detection procedures at the same time. With offloading procedure, each detection task can possess different execution time considering about the demo in last chapter. This can be caused by simple different determination of synthesis number of images or, more generally, by different significance level of virus.

Considering about the system resources limit. In order to execute more detections for different viruses. It is notable to label the importance of each virus for detection priority issue. With more important virus, more resources should be thrown in. Specifically in the project, detection period can be reduced to make a more frequent test. For those not significant viruses, a longer period can be adopted. In this way, system can pay more attention to real significant viruses detection. Another way to distinguish the detection is through image size. Likewise, for those viruses with smaller diameter, a larger image

can really help during the detection procedure while others with larger diameters can be resized with smaller size.

So for multiple application with offloading mechanism, multiple viruses can possess different sizes and of different image synthesis numbers during the detection procedure. As been talked before, the scheduling issue depends on corresponding parameters, which will be presented in next section.

5.2 Tool Build

This section will present an UI used for implementation of all the procedures introduced above. The interface is designed for yielding the best offloading decision based on relative inputs. The inputs parameters are already discussed in prior sections – number of tasks, number of versions for each task and corresponding characterized propertied such as *WCET*, *Deadline* and *Period*. In this section, the main procedures introduced prior will be scheduled and shown first to illustrate the data flow. Then in the end a Demo of interface will be presented.

5.2.1 data flow

For further illustration, data flow of the UI design will be introduced first. As been discussed before, it is clear that in the model of offloading mechanism application in virus detection project there will be different versions needed to be selected. In this case, characteristics such as period, execution time and deadline needed to be seen as input data. Also, as been discussed in last chapter, parameters value also needed to be considered here. In the demo, these characteristics will be send through algorithms in form of two dimensional matrix. Row data, for example, can be easily signed as different tasks with respect to different versions. Each row can be illustrated as $TnVi$ which stands for the i^{th}version of the n^{th} task. Column data, on the other hand, can be presented as characteristics of task: execution time, deadline and periods. As for value parameter, as it can be signed with respect to different definitions, it can be send through the algorithms with another matrix. Or it can be automatically determined during the calculation. One simple example is when dealing with offloading mechanism, the shorter programming time on the local embedded system, the higher benefit system can get, so

the value parameter can be easily signed as time difference between offloading version and non-offloading version – the original system. Under such circumstance, the value parameter can be easily signed through subtraction calculation between the first column of the input matrix – execution time and the original non-offloading version execution time.

Once the input data is send into the program, algorithms can be carried on to yield the final result. Apparently, in order to get the best selection number of task version, the approximated demand bound function must be yielded first. As been illustrated above, the function is modeled and determined by algorithm SMP (Selected Multiple Points). As been shown, the approximated demand bound function will be presented as three parameter P, Q and u. Again, the result of the function can be stored in a two dimensional matrix with row data versions of tasks and column data – P, Q and u. These parameters will be further processed.

Based on yielded approximated demand bound function parameters P,Q and u, associated with parameter value, result of best selection version can be got through algorithm HEU.

To summarize the whole data processing procedure, figure 5.1 is shown below. From the figure, it also can be seen the inputs and outputs of each algorithm.

As for specific realization approach and platform, next section – realization will be introduced.

5.2.2 realization

Since the project is based on platform Qt as introduced before, the UI interface will be also designed on platform of Qt. In case the UI interface needed to be carried on other platform, a *class* in programming language Cpp is defined to realize the whole processing procedures since platform Qt is based on Cpp. This is also one of the merits of the platform.

Specifically, since the input data needs to be send in in form of matrix, one *widget* named $Table$ in Qt is introduced for receiving the data. One reason of employing $QTable$ widget is it is more easier to receive the data as its resembling shape and characters – row

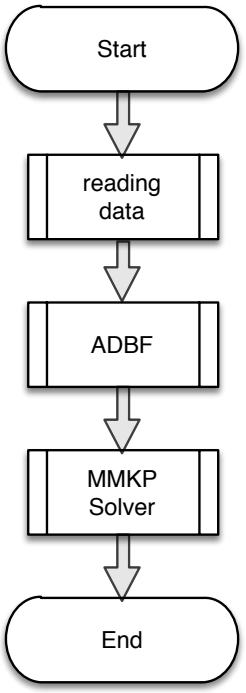


FIGURE 5.1: data processing procedure

and column. Another reason is as the number of tasks and versions are not determined first, it is more easier to define such widget characters – number of row in $QTable$ than other simple input widget such as $QLabel$ and $QTextEdit$. The number of tasks and versions is variable which determines the processing programming procedure will be all concerned with dynamic matrix definition.

As been discussed, the dimension of input parameters P, Q and u will be determined by predefined task number and version number. Yet these numbers should differ along with specific task models. For given offloading mechanism model, however, they are simply determined as follow: given task τ^i which $i = 1, 2..., n$; each task possessed version number w. Although the version number seems to be different with respect to the specific task τ^i, it would be more easier to signed with the same size considering about the following dynamic programming. Also one simple solution for scenario – tasks with different version numbers is to copy one prior version with respect to the corresponding task until all the tasks possess the same maximum number of versions. Theoretically it would not change the final result. To receive these two integer numbers, one Qt widget $QTextEdic$ is employed, a simply trick is since all data types received from Qt widgets are either $QString$ or $cell$, functions such as $toInt()$ should be implemented for data

type transform. For convenience of programming, class variable $TaskN$ and $VersN$ will be defined as number of task and number of version respectively which can be easily used for later variable definition such as determination of row number and column number of input matrix.

After receiving the number of task and version, a widget $PushButton$ is used for inputing data which values the parameter $TaskN$ and $VersN$, also, determine the row number of input matrix through setting characteristic $RowCount$ of widget $QTAble$. Again, after typing in all required input data – execution time, deadline and period of each task and version, $PushButton$ "Calculation" should be triggered with simple push to yield the final result.

In order to make a clear illustration, the explanation will follow the data flow presented in figure 5.1. Again to simplify the programming procedure and make it more transportable, all algorithms and processing procedures will be defined as class function and can be called and revised easily.

First, in order to carry on the further algorithms, one basic processing procedure should be employed – get jumping points from the inputing data as all later processing procedure will based on $ADBF$ whose parameters are determined by those jumping points. As been shown before, these jumping points will be yielded at time points which can be defined as below:

$$\sum_{i=1}^{n} C_i + mT_i \tag{5.1}$$

which i stands for task τ^i and m stands for number of period which can be signed as $m = 1, 2, ...$ With the equation above, it is not difficult for us to calculate jumping points of demand bound function. Meanwhile, for general demand bound function calculation, it is easy to yield the function within given time. In this case, however, it can be noticed that since the number of task and number of version varies along with the specific model, the number of task set combined with alternative versions with respect to tasks can be really big since it possesses an exponential growth along with the growth of number of task and version. Suppose 10 task is considered with offloading mechanism in the project, each task possesses 10 different versions, then the combination number should

be 10^10 which really makes the algorithm a bad processing procedure and can be seen as a NP problem. Who to solve the problem then?

In the project, one class function named $result\_dbf()$ is used for calculation of jumping points of demand bound function. The trick is instead of calculating demand bound function of whole task set, only one task one version is considered first. The combination part can be processed later in algorithm HEU. Briefly speaking, in the end of the function, the jumping points of each task version can be stored for following processing procedures.

After getting jumping points, one class function $result\_adbf()$ is used then to process the inputing data and yield corresponding result as been illustrated in figure 5.1 before. Specifically, in order to get the approximated demand bound function with inputing data, forementioned algorithms will be used here. As been discussed in last section, three parameter P, Q, and u should be valued to present the approximated demand bound function. For specific realization, the algorithm is designed to avoid exponential growth NP problem which caused by calculating combination of task versions, instead of simple straight forwards approximation of whole task set, $ADBF$ is concerned with simple one task one version, as been discussed before. This makes the calculation more easier as for one task, demand bound function will be more regular which makes the approximation more easier. As been shown in last section, yet according to the algorithm, the parameter u should be determined based on both parameter of P, Q and jumping point of demand bound function. For simple one task DBF, however, it should be simply defined based on jumping points since the shape of demand bound function repeats itself in every period which can be easily illustrated through figure 5.2.

As been shown in figure above, it can be easily seen that in the no matter where point (P, Q) is, the u value needs to be parallel with slope of the jumping points in DBF as the $ABDF$ should guarantee to possess a larger "demand" than corresponding DBF which means the approximated line should always above the original line and should not interact with each other. In conclusion, the value of u can be directly determined through the jumping points.

As for parameter P and Q, as been defined in prior section, in algorithm SWP, P point will be determined through H points chosen through trail and error which makes the programming efficient. In the programming, however, it can be done with simple setting

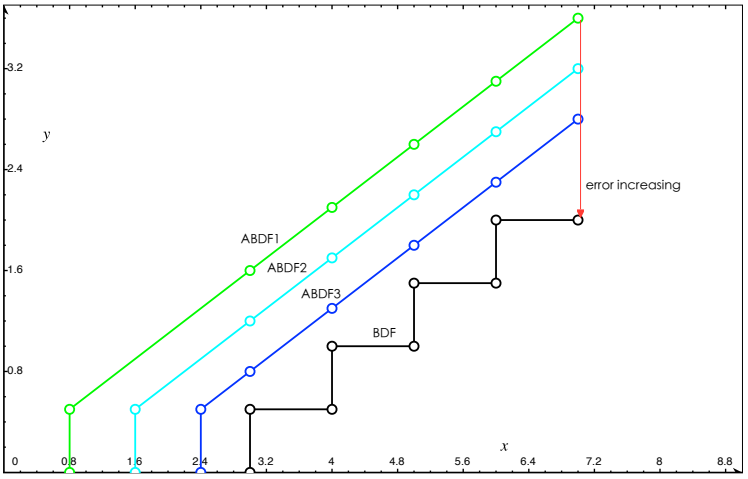


FIGURE 5.2: error illustration in ADBF

by staring jumping point of each demand bound function. Specifically, the aim of introducing H points is to make an approximation such that the starting point of $ABDF$ can be less than original demand bound function. Also the error between these two starting points will influence the approximation error between DBF and $ADBF$, which should be as less as possible. Given these consideration, in the programming, it is one point before the original starting point that is signed to be the approximated demand bound function. The error, as been discussed before, can be measured through subtraction calculation between these two starting points. One optimal solution is choosing the exactly original starting points to make the approximation. After determination value of P, Q point can be easily signed with the result of demand bound function with respect to the first jumping point. These results – u, P, Q are stored in class parameters – $Parameter\_u$, $Parameter\_P$ and $Parameter\_Q$ in form of one dimensional matrix respectively.

Since the $ABDF$ is presented by parameters P, Q and u which has already stored in corresponding matrixes as result of class function $result\_adbf()$, it is time to select the best version with algorithm HEU. In the programming, algorithm HEU is realized based on the forementioned results. As been described in algorithm 2, some parameters should be dynamically defined as along with the predefined parameters number of tasks and number of versions such as solution vector ρ and resource vector r. One parameter m – system resource number is set to be 1 while so far, in the project, only time limit is taken into consideration. Also, value parameter should be already signed with corresponding definition. As been introduce before, in algorithm, resource limitation should be checked

to guarantee the schedulability first. Considering about the $ADBF$ schedulability test procedures mentioned in last section – checking the sum of utilization rate u and each H points – mentioned in equation 3.5 and equation 3.6. After the schedulability test, versions of non-compliance will be eliminated from the selecting pool. Also, as been seen in the algorithm 2, a *while* loop here is employed trying to select the best version which is defined as version that possesses not only the highest system utilization rate but also highest penalty value per system resource.

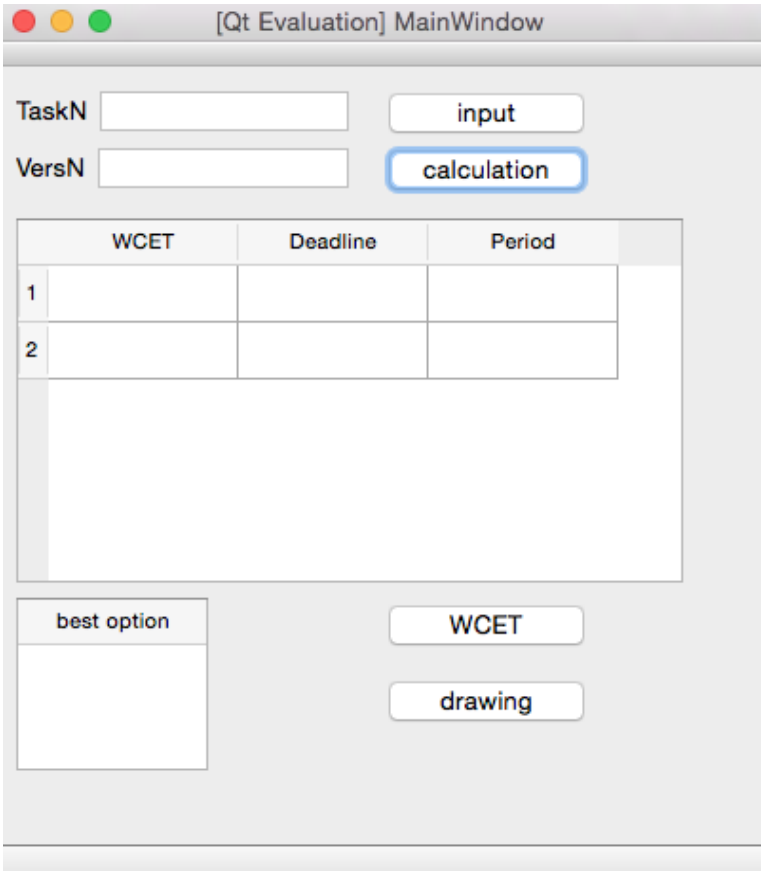


FIGURE 5.3: UI design for offloading decision poblem

In the end, the best solution yielded from HEU algorithm is send through Qt widget $QTable$ to be presented in the front window. Some data type transform should also be used here. One another functional push button "WCET" is also located in the front window which is designed to call other $WCET$ analysis software to yield the timing parameters directly after processing the source code. The known $WCET$ analysis tool software can be "aiT" from AbsInt, "RapiTime" from Rapita Systems or "Chronos" from National University of Singapore. The final result of UI design can be seen form above figure 5.3.

In the UI design, algorithms forementioned are implemented to solve the offloading decision problem. As for the model based on project virus detection, the UI design can be presented as figure above, it can present one general model of offloading problem. For other models, however, interfaces such as inputing data structure may vary along with the demand. Hence, some further work may still need to be done. One solution may lie in some *Qt* widgets. One widget *List*, for example, enables selection function which can be used as model selection before inputing and calculation. It makes the design more reliable and portable. Also, since the approximated demand bound function can also be realized with other algorithms and selection result can also be yielded employing other algorithms, new class functions can easily build based on corresponding algorithms and be used for addressing the problem with same class variable defined before.

5.3 variables

5.3.1 parameters

In order to schedule forementioned model with EDF algorithm, parameters should be measured first. As been shown above, different sizes of image and different image skipping number can influence the execution time of one task. So during the measurement, execution time has been measured with different size of images and different number of image skipping. The measurement result can be seen through following table. The order of viruses is concerned with the size of images, such different versions are caused by image cutting. As forementioned, due to the physical characters, candidates in the edging area are not sufficient enough for the detection, cutting edge area can be good method for resource saving. Also considering about the size of viruses, those with larger diameters can possess larger central area while those of small diameters may be cut off more.

diameter(nm)	*image size*	*3 images(s)*	*5 images(s)*	*7 images(s)*	*9 images(s)*
300	1080,145	0.412	0.480	0.478	0.473
250	1080,135	0.406	0.435	0.471	0.456
200	1080,125	0.412	0.435	0.457	0.455
100	1080,115	0.411	0.425	0.441	0.466
50	1080,110	0.384	0.429	0.435	0.440

TABLE 5.1: execution time for different viruses

As can be seen the execution time is measured through trail and error. Also, for different viruses with different diameters, different image sizes are determined. The rest columns imply the synthesis image number which can also lead to different execution time.

As for deadline and period, they can be set based on execution time. The following two tables illustrate the result of these two parameters respectively. All these parameters, as been discussed before, will be sent into the interface designed in last section, then the final result of optimal scheduling solution can be yielded. Another parameters – penalty and priority will be introduced later.

diameter(nm)	*3 images(s)*	*5 images(s)*	*7 images(s)*	*9 images(s)*
300	0.824	0.960	0.956	0.946
250	0.812	0.870	0.942	0.912
200	0.824	0.870	0.914	0.910
100	0.822	0.850	0.882	0.932
50	0.768	0.858	0.870	0.880

TABLE 5.2: deadline time for different viruses

diameter(nm)	*3 images(s)*	*5 images(s)*	*7 images(s)*	*9 images(s)*
300	1.648	1.920	1.912	1.892
250	1.624	1.740	1.884	1.824
200	1.644	1.940	1.828	1.820
100	1.644	1.700	1.764	1.864
50	1.536	1.716	1.740	1.760

TABLE 5.3: periodic time for different viruses

5.4 Penalty & Priority

As been mentioned before, in order to yield best solution, HEU algorithm is adopted for solving such MMKP problem. Then parameter penalty should be determined first. Such determination actually can be divided into two situations.

- **scenario a**: in this case, the penalty value increases when the demand of system resources decreases. In other words, versions demanding less system resources possess higher penalty. In this way, HEU algorithm will not be necessary as the best solution is the versions with least resource demanding. The typical example for such situation is signing the penalty as "system resource saving" parameter. As

it is the only concern in the project, then the less processing procedures executed in the local, the better the total penalty will be. In this project, more skipping images are encouraged.

- **scenario b**: in this case, the penalty value decreases along with the decreasing of system resources saving. In this scenario, when the demand increases, on one hand corresponding constrains may be violated, on the other hand penalty value will be higher. Such trade off situation is the reason of adopting HEU algorithm.

In the project, skipping procedure may increase the local execution time, system resources in the server, however, will be saved. This is because when one image is skipped, one execution period in the server will be saved. So although considering the fact that the less the skipping images number is, the better quality the result will be, skipping is still necessary. In the project, the penalty is determined with respect to the image skipping number. More images are skipped, more penalty will be yielded.

In last chapter, one priority based HEU algorithm is mentioned to reduce the computational complexity. In the project, such priority can be determined with respect to the importance of the viruses. Recalling the concept in priority based HEU, the tasks with higher priority will possess higher penalty in the end, which, in the project, will have more skipping images. So suppose viruses with less importance possess higher priority, that means more resources will be saving even through detection result quality may be influenced.

With all parameters forementioned, the optimal solution can be found through the designed interface easily.

Chapter 6

Conclusion

6.1 Conclusion

In the thesis, in order to implement offloading mechanism, several models have been proposed, result required from remote server, result require-less from the server, one task application, multiple tasks applications. Each of the models have been discussed and corresponding task scheduling problems have also been discussed. During which, multiple task based offloading mechanism model has been illustrated through specific one case study. The scheduling problem concerns problem such as Approximated Demand Bound Function and Multiple Multidimensional Knapsack Problem. Solutions such as HEU and priority based HEU have been proposed.

Through the solution, it can be noticed that with forementioned processing procedure, one schedulable solution can be yielded. Also, the solution can be effected with different determination result of parameter "penalty", which effects the iteration results of algorithm HEU. Also, the proposed priority based HEU algorithm reduces the computational complexity due to the signed priority. As for error reduction, the classification based approximated demand bound function is introduced. Finally, one general computation tool is built up to this kind of model.

With given model and computation tool interface, offloading mechanism can be implemented with multiple tasks which can be seen as schedulable and optimal solution based on the determined penalty. This offloading model can reduce the resource demand of task in embedded system. This seems more significance especially when "the free meal

is over". With such mechanism, more sophisticated algorithm can be adopted which means more complex functions can be realized now.

6.2 Prospect

As been shown, the offloading mechanism can be realized with different models. Here, only one specific model has been presented. Offloading model like result required task system, for example, still needs to be studied while self-suspension problem may arise. Also interface design can be improved when more models scheduling problems have been solved. WCET, for example, can be measured with softwares instead of measurement. More further works are needed.

Bibliography

[1] Ezio Malis. Survey of vision-based robot control.

[2] Giorgio C. Buttazzo. *Hard Tral-Time Computing System.* Springer, 2011.

[3] Pascal Libuschewski, Dominic Siedhoff, Constantin Timm, Andrej Gelenberg, and Frank Weichert. Fuzzy-enhanced, real-time capable detection of biological viruses using a portable biosensor. *Proceedings of BIOSIGNALS*, pages 169–174, 2013.

[4] Miller BP Powell ML. Process migration in demos/mp. In *ACM SIGOPS Oper Syst Rev*, pages 110–119, 1983.

[5] Yung-Hsiang Lu Bharat Bhargava Karthik Kumar, Jibang Liu. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1): 129–140, 2013.

[6] Balan RK. Powerful change part 2: reducing the power demands of mobile devices. In *IEEE Pervasive Comput*, pages 71–73, 2006.

[7] Kandemir M Vijaykrishnan N Irwin MJ Chandramouli R Chen G, Kang B-T. Studying energy trade offs in offloading computation/compilation in java-enabled mo- bile devices. In *IEEE Trans Parallel Distrib Syst*, pages 795–809, 2004.

[8] F. Ridouard. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 47 – 56. IEEE, 2004.

[9] James W. Layland C. L. Liu. *Journal of the ACM (JACM)*, volume 20. ACM New York, 1973.

[10] J. Lehoczky. The rate monotonic scheduling algorithm: exact characterization and average case behavior. *Real Time Systems Symposium*, pages 166 – 171, 1989.

[11] C. Krintz S. Gurun, R. Wolski and D. Nurm. On the efficacy of computation offloading decision-making strategies. In *Int. J. High Perform. Comput*, 2008.

[12] Y.-H. Lu Y. Nimmagadda, K. Kumar and C. Lee. Real-time moving object recognition and tracking using computation offloading. In *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference*, 2010.

[13] C. Liu and J. Anderson. Task scheduling with self-suspensions in soft real-time multiprocessor systems. In *Proceedings of the 30th RealTime Systems Symposium*, 2009.

[14] A. Messer I. Greenberg X. Gu, K. Nahrstedt and D. Milojicic. Adaptive offloading inference for delivering applications in pervasive computing environments. In *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 107–114, 2003.

[15] C. Wang Z. Li and R. Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *CASES*, pages 238–246, 2001.

[16] C. Wang Z. Li and R. Xu. Task allocation for distributed multimedia processing on wirelessly networked handheld devices. In *In Parallel and Distributed Processing Symposium., Proceedings International, IPDPS*, 2002.

[17] K. Yang S. Ou and A. Liotta. An adaptive multiconstraint partitioning algorithm for offloading in pervasive systems. In *In IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 10–125, 2006.

[18] S. Ou K. Yang and H.-H. Chen. On effective offloading services for resource-constrained mobile devices running heavier mobile internet applications. In *IEEE Communications Magazine*, 2008.

[19] Y.-H. Lu K. Kumar, J. Liu and B. Bhargava. A survey of computation offloading for mobile systems. In *Mob. Netw. Appl.*, 2013.

[20] A. Toma and J.-J. Chen. Computation offloading for frame-based real-time tasks with resource reservation servers. In *In the 25th Euromicro Conference on Real-Time Systems (ECRTS 2013)*, 2013.

[21] M. Spuri and G. Buttazzo. Efficient aperiodic service under earliest deadline scheduling. In *In Real-Time Systems Symposium*, 1994.

[22] M. Spuri and G. Buttazzo. Scheduling aperiodic tasks in dynamic priority systems. In *Real-Time Systems*, 1996.

[23] A. Toma and J.-J. Chen. Server resource reservations for computation offloading in real-time embedded systems. In *the 11th IEEE Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia 2013)*, 2013.

[24] S. Park D. Kim I. Kim, K. Choi and M. Hong. Real-time scheduling of tasks that contain the external blocking intervals. In *Proceedings of the 2nd International Workshop on Real-Time Computing Systems and Applications*, 1995.

[25] J. C. Palencia and M. Gonzlez Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *n Proceedings of the 19th IEEE Real-Time Systems Symposium*, 1998.

[26] R. Rajkumar. Dealing with suspending periodic tasks. In *Technical report, IBM T. J. Watson Research Center*, 1991.

[27] K. Tindell. Adding time-offsets to schedulability analysis. In *Technical Report 221, University of York*, 1994.

[28] C. Liu and J. Anderson. An o(m) analysis technique for supporting real-time self-suspending task systems. In *Proceedings of the 33th IEEE Real-Time Systems Symposium*, 2012.

[29] M. D. Natale H.Zhang. Using max-plus algebra to improve the analysis of non-cyclic task models. In *25th Euromicro Conference on Real-Time System*, 2013.

[30] C. Timmb F. Weicherta, M. Gaspara. Signal analysis and classification for surface plasmon assisted microscopy of nanoobjects. *Sensors and Actuators B: Chemical*, 151:281–290, November 2010.