2012

QoS and security-aware task assignment and scheduling in real-time systems

Naeem A. Al-Oudat
Iowa State University

QoS and security-aware task assignment and scheduling in real-time systems

by

Naeem Al-Oudat

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Manimaran Govindarasu, Major Professor
Doug W. Jacobson
Yong Guan
Daji Qiao
Sigurdur Olafsson

Iowa State University

Ames, Iowa

2012

To my parents, wife and children for their love, patience and support. Jazakom Allah Khaira.....

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

All praise is due to ALLAH, the most merciful and the most compassionate, for bestowing this Ph.D. degree among other endless blessings on me. Along the journey to make this dissertation, many people came to my assistance and support, and offered their love and compassion.

First and foremost, I would like to express my deep gratitude, appreciation, and thanks to Dr. Manimaran Govindarasu for his valuable assistance in pursuing my Ph.D. degree. Dr. Manimaran has guided me to this achievement with patience, vigor, research insight, and kind advice both professionally and personally. Without his help and advice, this dissertation could not have been possible.

Also, I would like to thank my Ph.D. committee members, Professors Doug Jackbson, Yong Guan, Daji Qiao, and Sigurdur Olafsson. They provided me with very helpful comments and feedback on my research proposal and dissertation, which greatly elevated the quality of my work and opened up future research directions for me.

As a member of the real-time systems and networking laboratory and the department of electrical and computer engineering, I benefited from the discussions, paper reviews, and presentations with my colleagues and friends. I would like to extend my gratitude to my lab mates Benazir Fateh, Kayal, Aditya Ashok, Adam Hahn, Mohammad Fraiwan, Siddharth Sridhar, Haribabu Narayanan, and Yoga Mahesh. Also, my deepest thanks go to my friends Ramzi Saifan, Jafar Kofahi, Bashsar Gharaibeh, Hisham Almasaeid, Sharhabeel Alnabelsi, Ahmad Tamrawi, Tamer Omar, Abdullah Almasoud and Mohammad Saleh.

In addition, I would like to acknowledge and thank my loving friends in the Ames and ISU Community. In particular, Hajim Bataina, Mohammad Al-saqer, Nimer Mehyar, Ehab Abu

Basha, Khalid Boushaba, Sa'Ud Al-Sa'Di, Omar Alajloni, Firas Shalabi and so many others that supported and put up with me all these years.

Finally, I am full of pride and gratitude to my parents, wife, children, sisters and brother for their love, effort, endless support, and help throughout my life.

ABSTRACT

Security issues in mission-critical real-time systems (e.g., command and control systems) are becoming increasingly important as there are growing needs for satisfying information assurance in these systems. In such systems, it is important to guarantee real-time deadlines along with the security requirements (e.g., confidentiality, integrity, and availability) of the applications. Traditionally, resource management in real-time systems has focused on meeting deadlines along with satisfying fault-tolerance and/or resource constraints. Such an approach is inadequate to accommodate security requirements into resource management algorithms. Based on the imprecise computation paradigm, a task can have several Quality of Service (QoS) levels, higher QoS result incurs higher computational cost. Similarly, achieving a higher level of confidentially requires stronger encryption, which incurs higher computational cost. Therefore, there exists a tradeoff between schedulability of the tasks on the one hand, and the accuracy (QoS) and security of the results produced on the other hand. This tradeoff must be carefully accounted in the resource management algorithms. In this context, this dissertation makes the following contributions: (i) formulation of scheduling problems accounting both deadline and security requirements of workloads in real-time systems, (ii) development of novel task allocation and scheduling algorithms for such workloads, (iii) and evaluation of the results through simulation studies and a limited test evaluations in one case. In particular, the following are the three key contributions.

Firstly, the problem of scheduling a set of non-preemptable real-time tasks with security and QoS requirements with the goal of maximizing integrated QoS and security of the system is addressed. This problem is formulated as MILP, and then its complexity is proved to be NP-hard. An online efficient heuristic algorithm is developed as the problem is NP-hard.

Simulation studies for a wide range of workload scenarios showed that the proposed algorithm outperforms a set of baseline algorithms. Further, the proposed algorithm's performance is close to the optimal solution in a specific special case of the problem.

Secondly, a static assignment and scheduling of a set of dependent real-time tasks, modeled as Directed Acyclic Graph (DAG), with security and QoS requirements in heterogeneous real-time system with the objective of maximizing Total Quality Value (TQV) of the system is studied. This problem is formulated as MINLP. Since this problem is NP-hard, a heuristic algorithm to maximize TQV while satisfying the security constraint of the system is developed. The proposed algorithm was evaluated through extensive simulation studies and compared to a set of baseline algorithms for variations of synthetic workloads. The proposed algorithm outperforms the baseline algorithms in all the simulated conditions for fully-connected and shared bus network topologies.

Finally, the problem of dynamic assignment and scheduling of a set of dependent tasks with QoS and security requirements in heterogeneous distributed system to maximize the system TQV is addressed. Two heuristic algorithms to maximize TQV of the system are proposed because the problem is NP-hard. The proposed algorithms were evaluated by extensive simulation studies and by a test experiment in InfoSpher platform. The proposed algorithms outperform the baseline algorithms in most of the simulated conditions for fully-connected and shared bus network topologies.

CHAPTER 1 Introduction

Real-time systems and their applications are widely used and spread in today's life. Examples of these applications include avionics, air traffic control, factory automation and defense (command and control). Real-time system is becoming pervasive, where more and more of the world's infrastructure depend on it [3]. Real-time systems range from simple standalone, e.g., digital camera, to more sophisticate and complex systems, e.g., agile manufacturing.

1.1 What is a Real-time System?

A real-time system is that computing system where its performance is dependent not only on correctness of the output but also on the timing of producing the output. Real-time systems have several attributes that are important for understanding and dealing with such systems:

- *Real-time tasks.* Real-time task has to be completed by a specific time called deadline. Real-time can be periodic (inter-arrival times are equal) or aperiodic (different inter-arrival times and only minimum inter-arrival times are known). Real-time systems are classified, according to tasks run on the system, to hard and soft real time systems. Delays in hard real-time system response result in a catastrophic consequences while delays in soft real-time systems have only degraded performance effect. Further tasks can be preemptable (stopped before completion in favor of another task) or non-preemptable (once started runs to completion).

- *Task scheduling.* Scheduling decisions in real-time computing systems have to be efficient in assigning available resources to tasks in order to meet the timing constraints of the tasks. A number of scheduling algorithms have been proposed and studied in the litera-

ture for real-time systems, e.g., EDF, RM, etc [4]. Each of these algorithms has its own advantages and disadvantages in terms of schedulability (number of guaranteed tasks), computation overhead, resources utilization and response to dynamics of the system.

- *Real-time architecture.* Real-time systems can be classified, according to its structure, into a uniprocessor, a multiprocessor or a distributed system. Uniprocessor and multiprocessor systems execute tasks on one node. Distributed system, on the other hand, consists of several computing nodes that are connected to each other via a communication network. Distributed nodes cooperate to achieve a common goal by executing the applications in the system.

1.2 Distributed Real-time Systems

A distributed system consists of several computing nodes that are connected to each other via a communication network. The distributed nodes cooperate to achieve a common goal by executing the applications in the system. To guarantee timing constraints in such systems a deterministic communication protocols should also be employed for underlying network (e.g., RTP [5]). Besides the operating systems that are running on each of the nodes in the system there are several middleware systems (distributed systems) that can be installed on the distributed nodes. A middleware refers to the set of services composed of IAA (Identification, Authentication and Authorization), APIs (Application Programming Interfaces), and management systems which support the needs of a distributed, networked computing environment [6]. The list of middleware systems includes:

- RCES4RTES (Reconfigurable Computing Execution Support for Real-Time Embedded Systems) [7] middleware. It supports DRE systems.

- DynamicTAO [8] is an extension of the TAO middleware [9] to support adaptive applications running on dynamic environments. DynamicTAO implements concurrency, security and monitoring mechanisms; and the dynamic migration, loading/unloading of components at runtime.

- The Fault-tolerant Load aware and Adaptive middlewaRe (FLARe) [10] extends TAO and supports DRE systems. FLARe is an efficient QoS-aware component middleware.
- The Component Integrated ACE ORB (CIAO) [11] is a free implementation of Real-Time CORBA specifications [12].
- SwapCIAO [13] is an extension of CIAO middleware to support reconfigurable DRE systems.
- PolyORB HI [14] is a minimal middleware core that provides common services for the applications.

1.3 Security-sensitive Real-time Systems

In many security sensitive Real-Time Systems, it is important to guarantee the security (e.g., confidentiality, integrity and authentication) and highest quality (e.g., high resolution, more color depth, or high rate) of exchanged information, while meeting the timing constraints. Systems with these features and capabilities are widely used, e.g., in industry and military applications like factory automation, smart power grid and battle field vision systems [15, 16]. Despite a possible usage of the Security Threat Estimators (STEs) to warn the RTES system of any probable security breaches, the threat cannot be exactly assessed. The warning messages from STE can be related to the lowest acceptable security level that should be used to encode transmitted messages over the network. Given the lack of exact threat assessment, the security provisions used in the system should be as high as possible, [17–19], considering the risk level as a lowest acceptable level.

The challenge in this context is how to efficiently execute applications in these mission critical real-time systems while meeting non-functional requirements, such as timeliness, security, robustness, dependability, performance etc. This is where QoS management applies. QoS-aware applications have an important property; they can perform at degraded levels and still provide a satisfactory result to a certain degree of accuracy.

In many real-time systems, tasks arrive dynamically to the system where non-preemptive scheduling policy is preferred because it is deadlock free and has low overhead [20]. For a team of Unmanned Aerial Vehicles (UAVs) that is deployed in a hostile area to capture images of some targets for analysis purpose, the security, image quality and number of images are important. Considering one UAV at a time, the tasks (e.g. image processing jobs) in this system (UAV) should be run in the highest possible levels of security and QoS while the system is not overloaded. As the system load increases (i.e. more targets appear in the area), security and QoS levels of the running tasks should be modified to give a room for the new tasks, while meeting the timing constraints of the admitted tasks. Creating room for the arriving tasks implies modification of the execution times of admitted tasks. Execution times are closely connected to the security and QoS levels of the task, therefore some tasks will be the subject of QoS and security degradation to acceptable levels that meets security and timing constraints. On the other hand, when the system load decreases due to the departure of some tasks (i.e. target moves out of sight), the system should increase the QoS and security levels of some tasks to improve the performance (in terms of QoS and security). Efficient decision on what tasks and what levels of QoS and security are the subject of the modification is not a trivial task to do, especially if the goal is to maximize the system utility.

More involving allocation decisions arise when the application is a group of tasks that has precedence relationships and have to communicate some data to each others. This kind of application subtasks can be seen as more required processing stages of the captured images by the team of UAVs. Because of the nature of this load, which can now be represented as a task graph, a cooperative execution of applications is possible by careful assignment of application parts to different sites (UAV nodes). The allocation of applications to several nodes involves application selection, task selection within the application, site selection where the task should be assigned and QoS level selection. QoS maximization can be considered at some or all of these points while security encoding/decoding overhead of the data exchanged between communicating subtasks should be accounted for.

1.4 Thesis Statement

Security issues in mission-critical real-time systems are becoming increasingly important as there are growing needs for satisfying information assurance in these systems. QoS needs of applications in these systems brings a tradeoff between schedulability of the tasks on the one hand, and the accuracy (QoS) and security of the results produced on the other hand. Design and evaluation of a dynamic scheduling of a set of independent tasks with QoS and security requirements in uniprocessor system to maximize QoS and security of the system is provided. Design and evaluation of a static allocation of a set of DAGs with security and QoS requirements in distributed real-time system is considered. Design and evaluation of a dynamic allocation of a set of DAGs with security and QoS requirements in distributed real-time system is considered.

1.5 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2, the details of system architecture used in this dissertation are given and discussed. Chapter 3 gives the details of single uniprocessor node scheduling algorithm. The details of static scheduling algorithms on distributed heterogeneous sites are given in Chapter 4. In Chapter 5 a discussion of the dynamic scheduling algorithm in distributed systems is provided. Conclusions and some future work directions are stated in Chapter 6.

CHAPTER 2 System Architecture

In this chapter, the system model, application scenario, related work and identified research problem are provided. In Section 2.1, system model and its major components are discussed. In Section 2.2, a scenario of an application that may benefit and use our research results is identified. Thesis contributions are identified in Section 2.3.5. Some related research is discussed in Section 2.3.

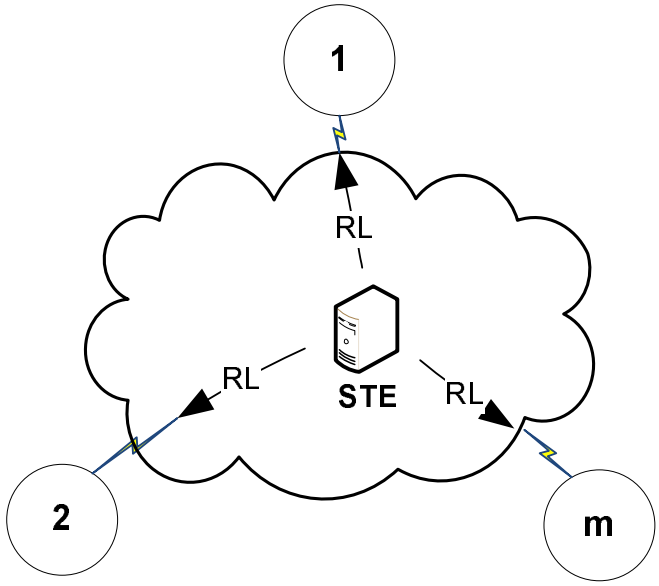


Figure 2.1 System Model

2.1 System Model

The system under consideration is a real-time computing system that employs real-time scheduling algorithms, e.g., EDF. Fig. 2.1 shows the system model considered in this dissertation. The system consists of several nodes up to m that are connected to each other via

a public network. Hence, a security infrastructure (Security Threat Estimator: STE) is presumably deployed to monitor the underlying network for any potential security breaches. STE warns all parts of the system about any security breaches in a form of Risk Levels (RLs).

There are three major components that are interrelated and compile the system which is studied in this dissertation. These components are; sites, tasks and allocation decisions.

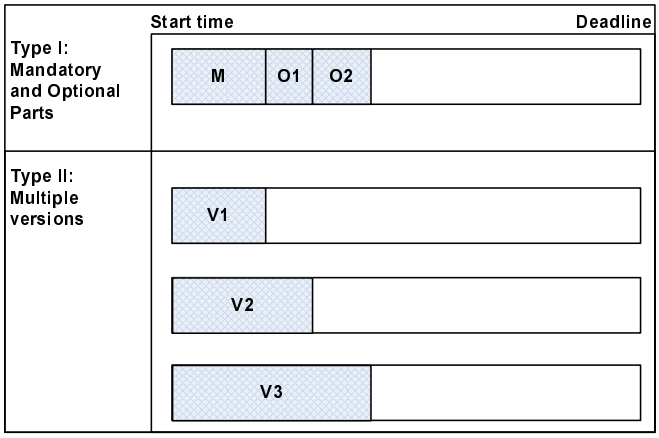


Figure 2.2 A Configurable Task Model

- *Sites.* System can consist of single site or distributed sites. Single node or site can be part of larger system that is connected by any type of networking. However single site has its own resources and constraints that are not affected by other parts of the system and further it does not cooperate with other sites to execute any tasks. On the other hand distributed system consists of several sites that are cooperating to achieve a common goal. Sites in distributed system can be identical (homogeneous) or heterogeneous in their computational and resource capabilities.

- *Tasks.* Tasks considered in this dissertation are configurable tasks. Imprecise computation was introduced by Lin et al. [21], where a real time task may have two or more versions that do the same job with different execution times and accuracies of the results, see [22], [23], [24], [25]. Or the task can consist of two parts: Mandatory part and optional part. If the task has time to complete both parts the result is said to be precise; whereas the task result is said to be imprecise, if not all parts are fully completed. Hence the

optional part refines the result of the mandatory part. If the optional part is divided into several portions the level of refinement and hence the level of accuracy is proportional to the number of completed optional portions. An Example of both types is shown in Fig. 2.2.

Tasks can be independent where each task is a standalone computational block or dependent where tasks are composed of interconnected subtasks. A dependent task is modeled as a task graph called Directed Acyclic Graph (DAG) where edges represent precedence and dependency relationship between tasks.

Non-preemptable task is not interrupted once start execution while preemptable tasks can be interrupted by other tasks that has a higher priority. Although preemptive scheduling can achieve high system utilization, preemption can be in some hardware or software configuration impossible or expensive [26]. Non-preemptive scheduling policy, on the other hand is deadlock free and has low overhead [20], [27], [28], [29]. Therefore non-preemptable tasks are used in most parts of this dissertation.

- *Allocation decisions.* Allocation means assignment of tasks to sites and scheduling these tasks on the specified sites. The type of allocation decision is *static* or *off-line* when application's parameters and resources during the whole run of the system are known beforehand. Allocation type is considered *dynamic* or *on-line* when allocation decision has to be taken during system run. Dynamic allocation decision has to be used when applications and resources available vary during the course of the system run.

2.2 Application Scenario

As an application scenario, consider a surveillance team of Unmanned Aerial Vehicles (UAVs) deployed to detect and classify targets in a battlefield area [1]. The UAVs are heterogeneous in computational capabilities and process the data from several local inputs to extract flight parameters and target information besides data received from other team members (see Fig. 2.3). Some of the processing blocks (e.g., guidance and control) shown in Fig. 2.3 should be

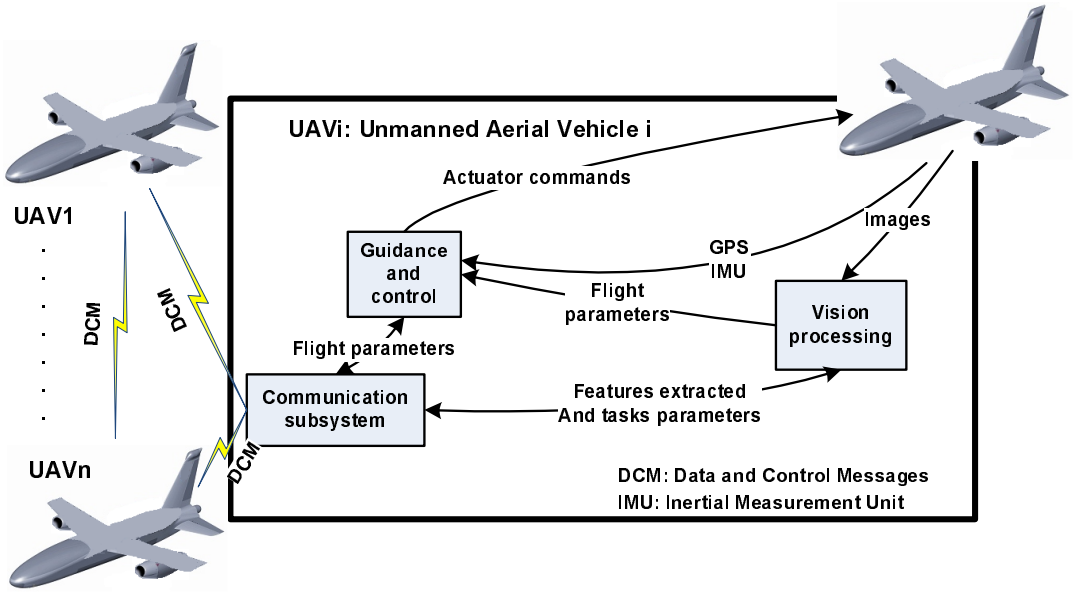


Figure 2.3 Cooperating UAV team schematic diagram [1]

totally executed on the same UAV board. Vision processing is one of the key processing blocks, which is used to process the captured images (i.e., extracting features, classifying targets, etc), and is also the most computationally intensive block.

The environment calls for security provisions (Confidentiality, Integrity and Authentication) of the collected data. In this scenario, the threat level is pre-estimated by a security threat estimator (STE) based on the mission and on the likelihood of an attack, and can be updated while the UAV is in action. Based on the expected number of targets and the available number of team members, the job assignment and scheduling are conducted offline. Then, during the course of the mission (at run-time), the actual number of targets can be different and hence the number and nature of jobs calls for an online allocation of the new load on the available resources.

A vision processing task is composed of several subtasks and can be modeled as a DAG. Fig. 2.4 shows the details of a target detection/classification task. The captured image is split into several segments based on the number of targets, and then after classifying targets on each of the segments a route change or/and any similar decisions are taken. Notice that each block in Fig. 2.4 can have several versions that implement the same functionality with different

execution times and hence different accuracies (QoS).

Further when a UAV node is not able to cooperate with other UAVs in the team, the scheduling decision should be taken locally to respond to load fluctuations. Degrading accuracies for some of the tasks can be one of possible scheduling decisions that take place on UAV board.

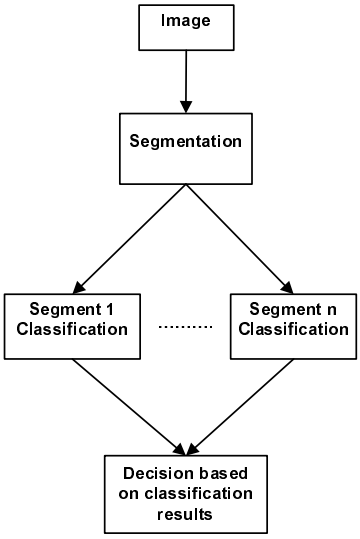


Figure 2.4 Target recognition application flow

2.3 Related Work

Related research can be classified into several categories according to addressing of QoS and security issues or to using of tasks modeled as Directed Acyclic Graph (DAG):

2.3.1 Security Issues in Real-time Systems

Security issues in the real-time systems have been addressed in the literature and discussed along with the schedulability of the system [17, 19]. Xie et al. [30] used local optimization of the security level to maximize the guarantee ratio (the ratio of the completed to accepted number of tasks). The tasks in queue, with lowest execution time, are given the highest security level. In [17], the authors optimized security as strength of defense metric, which is the average of normalized security levels of all tasks. They aimed at maximizing the guarantee

ratio at all working conditions. During the overload conditions, the security level of a preset number of earliest deadline-tasks is decreased. In [19], the authors proposed SASES algorithm to maximize the security value of the overall system while meeting the time constraint by choosing the task that maximizes the benefit to cost ratio (security value to computation). In [31], the authors proposed a group based security model. Each group has some security services that provide the same security type with different qualities and different computation times. They chose security services, one from each required group according to quality and overhead (computation time) to achieve the maximum system wide combined security while maintaining schedulability and minimum computation requirement.

2.3.2 QoS in Real-time Systems

QoS has also been addressed in [26,32]. The authors in [32] model QoS as a reward for the contract between the server and clients, where the server tries to maximize its reward from all contracts. This could result in losing some contracts under overload conditions, but the server has the right to choose what client to drop so as to allow graceful degradation of the system performance in terms of the reward. In [26] QoS is used in developing QoS-aware fault tolerant scheduling algorithms for real-time heterogeneous clusters.

2.3.3 Security and QoS in Uniprocessor Real-time Systems

Kang and Son [18] tried to optimize QoS, while keeping the system security level greater than the risk level during the operation of the system. They achieve this by computing all possible combinations of security levels and QoS levels of the task set off line and choose from this list during risk level perturbation. In [33] the authors proposed a framework for maximizing the utility function of the system from scheduling a set of tasks with dependent multidimensional QoS levels (i.e. accuracy, reliability, cryptography, etc.) on multiple resources. They tried to maximize the system utility in a way where some of the QoS dimensions might not improve beyond its minimum value. They extended this work to support a dynamic task traffic model [34].

2.3.4 Security and QoS in Distributed Real-time Systems

In their work in a distributed system Rajkumar et al. [35] considered maximization of the system utility from allocating independent preemptable tasks in a distributed system. They further improved their techniques to reduce the computation complexity of the initial proposal and applied it to radar tracking in [36].

2.3.5 Directed Acyclic Graphs (DAGs) Allocations in Distributed Systems

Suitable applications for distributed systems are modeled as task graphs that are called Directed Acyclic Graphs (DAGs). Allocation of applications on a set of distributed processors has been extensively studied by the community. Two general methods are used to assign DAGs to distributed processors. First method is clustering based where tasks are clustered according to some criteria [37], [38]. Then assignment to actual sites is taken place. Second method is the list based assignment where tasks are given priorities according to its importance in the graph. The task with highest priority is considered for assignment first. After assignment the scheduling stage is taken place in both assignment methods.

Ramamritham used the clustering method in [38]. He introduced a clustering method where tasks are clustered according to the ratio of the communication cost to the computation cost of the communicating pairs of tasks. He takes into account period of the DAGs as a deadline and communication between nodes when doing the allocation.

Starvinides and Karatza [39] used a list assignment policy. They addressed the effect of error in input to components of an application modeled as a DAG after partially completed preceded components, making benefits from imprecise computation idea in [21]. They proposed a modified versions of well known algorithms (i.e., EDF, LSTF, and HLF), to dynamically allocate tasks on a homogeneous distributed real-time system.

Recent survey [40] gives some discussion about real-time DAGs scheduling on multiple processors. The survey [41] gives a detailed discussion of DAGs assignment and scheduling in distributed systems. A recent paper [42] studies several DAG scheduling algorithms on multiple processors and makes a comparison between the studied algorithms that are classified

into several groups. Their comparisons are based on the speed up, makespan, schedule length ratio, and processor utilization. In this dissertation, dependent tasks are modeled as DAGs and a modified version of list scheduling is used.

Resources allocation has received a lot of attention in operations research field. The resource-constrained project scheduling problem (RCPSP), [43], with resource allocation is similar to the problems of DAG allocation in distributed system considered in this dissertation. RCPSP consists of activities that must be scheduled subject to precedence and resource constraints such that the makespan is minimized. RCPSP generalizes the known job-shop scheduling problem [44] and it is NP-complete.

Other related works from operations research field are in the area of reservation systems to admit as many customers (jobs) as possible to the system such that the system utility (e.g., end-to-end delay, fairness, makespan) is optimized. Examples include downtown space reservation system citezhao2010travel, and admission control in single-hop multiservice wireless networks [45]. Yield management for airlines, hotels, broadcasting advertisements, and car rentals where customers arrive arbitrary to the system represents the dynamic version of reservation system scheduling problems. Yield management problems are special cases of multidimensional knapsack problem [46].

In summary, there is not much research in the literature that considers scheduling of tasks with QoS and security requirements. For those researchers who addressed both dimensions, they have considered independent preemptable tasks and their goal was to maximize only one of the dimensions.

Assessment of security threats in the underlying network is only an estimate; therefore the security provisions in the system should be as strong as possible. QoS should also be maximized. Therefore an integration of both security and QoS should be maximized. However, to the best of our knowledge there is no research that considered the balance in QoS and security for non-preemptable tasks with QoS and security requirements.

Although most real-time applications are in the form of dependent tasks, to the best of our knowledge there is no research that considered these types of applications with QoS and

security requirements for allocation in distributed real-time systems.

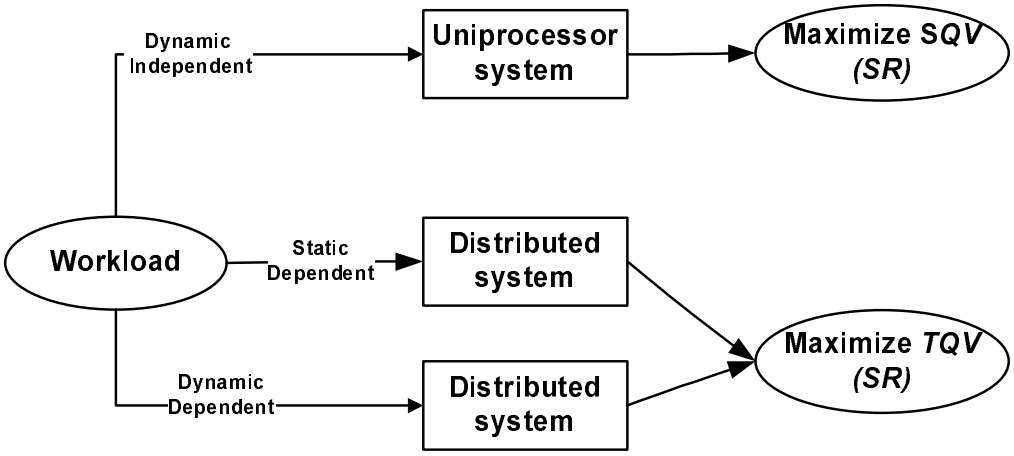


Figure 2.5 Thesis Contributions

2.4 Thesis Contributions

Three of the problems that are raised from the integration of the above mentioned components are identified and studied in this dissertation, see Fig. 2.5:

- Dynamic scheduling of independent tasks on uniprocessor site. While static allocation of independent tasks on uniprocessor with the goal of maximizing security and QoS is studied in the literature [18], there is no research that tackled the scheduling of dynamic load on such system. This problem is identified and studied in Chapter 3.

- Static allocation of dependent tasks on heterogeneous distributed system. Workload tends to be a form of dependent tasks in distributed real-time systems. When a preliminary decision on tasks allocation has to be taken, static allocation comes to picture. In Chapter 4 static allocation of tasks on distributed system is identified and studied.

- Dynamic allocation of dependent tasks on heterogeneous distributed system. In reality most of the workloads are dynamic and resources need to be allocated to them online. In Chapter 5 the details of dynamic allocation part are given. A proof of concept, using simple experiment on InfoSphere platform, is also provided in Chapter 5.

CHAPTER 3 Dynamic Scheduling in Uniprocessor System

3.1 Summary

In this chapter, the issue of scheduling a set of dynamic, independent, and non-preemptable real-time tasks in a uniprocessor system with specified QoS and security requirements is addressed. In particular, an approach to balance this tradeoff is developed, and the following contributions are provided: (i) formulation of a general non-preemptive real-time scheduling problem to maximize Security-QoS Value (SQV) is provided and (ii) an online heuristic algorithm called *SQV_EDF*, based on the Earliest Deadline First (EDF) scheduling for the problem is developed as it is NP-hard. Simulation studies for a wide range of workload scenarios showed that SQV_EDF achieves a higher performance than that of the MIN_EDF and MAX_EDF baseline algorithms, in terms of SQV and number of guaranteed tasks. In addition, it is shown that when the tasks are preemptable and have the same arrival time, the SQV_EDF algorithm is able to achieve a SQV performance closer to that of the optimal solution obtained by solving ILP.

3.2 Background

For a team of Unmanned Aerial Vehicles (UAVs), Fig. 2.3 that is deployed in a hostile area to capture images of some targets for analysis purpose, the strength of security used to decode the data transmitted to the ground station, image quality and number of images are important as per each node of the team. Assuming the communication between team members is not possible such that each node has to completely process its captured image and send it to the ground station using wireless media. Therefore the system requirements in this case; security,

QoS and number of successful jobs (equivalent to number of images), have to be maximized. In such hostile environment the estimated risk level (using suitable security infrastructure; STE: Security Threat Estimator) can also be fluctuating during the course of the team mission. As a result of this fluctuation, load on the system may increase due to more needed time to support high security level. Hence some of the jobs that cannot support a security level equal to or higher than the risk level should be terminated to keep the system secure. Therefore the system should be able to adapt to fluctuations in the risk level and modifies tasks parameters on the fly. Any terminated job results in a new room at that node that can be used to raise the QoS or/and security of other tasks in the system.

Fluctuation in the load (number of jobs) is also another dimension of this kind of dynamic systems. The tasks in each node should be run in the highest possible levels of security and QoS while the system is not overloaded. As the system load increases (i.e. more targets appear in the area of UAV team), security and QoS levels of the running tasks should be modified to give a room for new tasks, while meeting the timing constraints of the admitted tasks. Creating room for the arriving tasks implies modification of the execution times of admitted tasks. Execution times are closely connected to the security and QoS levels of the task, therefore some tasks will be subject of QoS and security degradation to acceptable levels that meets security and timing constraints. On the other hand, when the system load decreases due to the departure of some tasks (i.e. target moves out of sight), the system should increase the QoS and security levels of some tasks to improve the performance (in terms of QoS and security). Efficient decision on what tasks and what levels of QoS and security are subject of the modification is not a trivial task to do, especially if the goal is to maximize the system utility.

In this chapter, the problem of online scheduling of non-preemptable real time tasks with QoS and security requirements on a single processor with the objective of maximizing both QoS and security strength of the system is addressed. Given that the basic problem of online non-preemptive scheduling of tasks is NP-hard in the strong sense [47], the problem at hand is also NP-hard. We formulate this problem as a Mixed Integer Linear Program (MILP)

optimization problem and then we propose a heuristic algorithm (SQV_EDF) to optimize the performance of the system in terms of QoS and security levels of the transmitted information. The proposed algorithm tries to keep QoS and security levels of a given set of tasks in their best values while responding to the system workload fluctuations. It is very important for node to be responsive to the threats of the underlying network reported from STE by modifying the security levels of the tasks as the risk level varies, while maintaining graceful degradation of the system performance.

To the best of our knowledge this is the first dynamic non-preemptive real time scheduling algorithm on a single processor for set of tasks with security and QoS requirements, which maximizes the integrated security and QoS value of the system.

The rest of this chapter proceeds as follows. In Section 3.3, the related work on RTES security and QoS optimization is reviewed. Section 3.4 discusses the system model and gives an application scenario. The problem definition is presented in Section 3.5. Section 3.6 gives the details of the proposed algorithm. An illustrative example is given in Section 3.7. Simulation results are presented in Section 3.8. Conclusions are presented in Section 3.9.

3.3 Related Work

QoS and Security issues in RTESs have been addressed in the literature and discussed along with the schedulability of the system [17–19, 30–32]. In [48], the authors proposed a heuristic algorithm to maximize the system QoS while maintaining energy and schedulability of the system. System QoS resulted from running each task in one of several available modes (frequency, release time or execution length) with its respective quality. Also in [49], the authors proposed a procedure to reduce the system overall energy consumption with window-constraints guarantee. Their approach consists of two phases; off-line phase where tasks are guaranteed with their mandatory parts then in on-line phase the QoS is modified to respond to run time dynamics, by adding/removing optional parts from admitted tasks.

In [18] the authors proposed an algorithm to optimize QoS, while meeting the system security constraint. security level should always be greater than the risk level. They achieved

this by computing all possible combinations of security levels and QoS levels of the task set offline and choosing from this list during risk level perturbation. The authors, in [33], proposed a framework for maximizing the utility function of the system from scheduling a set of tasks with dependent multidimensional QoS levels (i.e. accuracy, reliability, cryptography, etc.) on multiple resources. They tried to maximize the system utility in a way where some of the QoS dimensions might not improve beyond its minimum value. This is good for some of the dependent dimensions but not when the dimensions are independent. For independent dimensions (security and QoS) which are used in this dissertation, we assume that the change in QoS level does not increase the size of the message to be encoded in some security level. Therefore our goal is to maximize both security and QoS levels, which makes our work different from theirs.

In this chapter, dynamic, non-preemptable and periodic real-time tasks is considered. The idea of imprecise computation that was introduced in [50] is used, where each task is assumed to have a set of discrete optional parts that we refer to as QoS levels. A system with an admission controller and STE system for threat monitoring in the network is considered. Then an online heuristic scheduling algorithm to maximize the SQV (integrated security and QoS metric) is proposed.

3.4 System Model

The system model is shown in Fig. 3.1. In this model a group of UAVs exchange control and data messages with a Ground System (GS) through a wireless network. It is of paramount importance that the GS maintains an up-to-date picture of the system via the status messages it receives from the UAVs and issues timely control actions to keep the system functional.

Thus, a STE or a similar security infrastructure is required to monitor the network for any potential security threats. This system reports threats in the form of risk level (RL)), to the other components of the model. A preemptable low overhead task is running periodically on each component of the system to decode the alarm messages from STE and update RL). RL level j corresponds to security level j for a task to meet its security requirements.

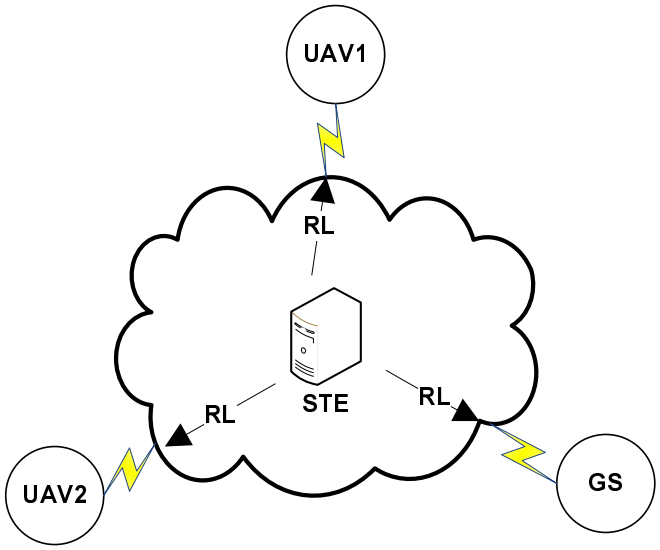


Figure 3.1 System model

3.5 Problem Formulation

In this section we provide the task model, performance metrics and problem statement.

3.5.1 Task Model

The tasks $\tau = \{T_1, T_2, \ldots, T_n\}$ in this system are periodic, independent, non-preemptable and run on one node. Each task is described by a 6-tuple $T_i = \langle C_i, R_i, Ql_i, Sl_i, D_i, P_i \rangle$, where C_i is the execution time of the real time task $i \in \tau$ without considering security (i.e., only first QoS execution time). R_i is the arrival time of task i, which is also considered as the ready time. Since tasks can arrive dynamically, the value of R_i varies for each task. R_i will not be used in any further calculations throughout this chapter. Given K_i security levels for a task i, Sl_i denotes the set of computation times for security levels, such that, $Sl_i = \{Exec\_S_i(k) : 1 \leq k \leq K_i\}$. Similarly Ql_i is the set of computation times for task i QoS levels, such that $Ql_i = \{Exec\_Q_i(l) : 1 \leq l \leq L_i\}$. D_i is the deadline of task i, which is assumed to be equal to its period (P_i). For the task i with QoS level l and security level k the current computation time C_i^c is given as $C_i^c = C_i + Exec\_S_i(k) + Exec\_Q_i(l)$. The current CPU utilization (U_i^c) by task i is given as $U_i^c = C_i^c / P_i$.

3.5.2 Performance Metrics and Parameters

The traditional metrics defined above are not enough. Therefore, we now define new metrics that we use in the algorithm.

- **QoS Metric.** The amount of time available for a task can be related to a QoS levels in the range $[1, 2, \ldots, L_i]$, where 1 and L_i are the lowest and highest levels for task i, respectively. Each QoS level, l, is given a weight, W_l^q; $0 \leq W_l^q \leq 1$, that reflects its relative importance among other levels. The intuition behind this is that the value returned from using a QoS level will not always increase linearly with the QoS level. The execution time (computation overhead) of the QoS level of task i is a function of the level number l, $Exec\_Q_i(l)$. We assumed the QoS levels, their respective computation times and weights are provided as task parameters. Note that higher the QoS level, higher is the computation time for a task. Providing a computation time for each QoS level can put an overhead on the application developer for such kind of tasks. This can be done by making use of milestone or sieve approaches [50]; or doing some QoS profiling as presented in [51].

  A higher QoS, for example, can mean higher accuracy (as a result of additional processing [52]) or better shape of the collected data (e.g. filtering, image compression). In order to capture the importance of a QoS level l for a task i in comparison with other tasks in the system, a normalized level of the QoS is used, denoted as Q_i^l, see Eq. (3.1).

$$Q_i^l = \frac{l}{L_i} W_l^q, \quad l = 1, 2, \ldots, L_i \tag{3.1}$$

- **Security Metric.** Security level efficiently represents various dimensions of security such as confidentiality, integrity and authentication. Each security level includes some or all of the security dimensions, where the highest level includes the strongest of each of the three dimensions. The integration of security dimensions into levels is beyond the scope of this dissertation. We restrict ourselves to encryption dimension of the security, where the strength of encryption is proportional to the key length used assuming the same algorithm is used system wide. Hence the level is mapped to a key length, e.g., level

1 can be an encryption/decryption with key length of 32 bits while level 2 represents encryption/decryption by a key of 64 bits long assuming a common block encryption algorithm is used.

Security levels are in the range $[1, 2, \ldots, K_i]$, where 1 and K_i are the lowest and highest levels. The security level that is used by any task should be greater or equal to the risk level sent by the STE to the system. Each security level, k is assigned a weight, W_k^s; $0 \leq W_k^s \leq 1$, that reflects the relative value returned from using such level. The security overhead (execution time), $Exec\_S_i(k)$, of task i increases as the security level k increases [17]. The computation times of the security levels and importance weights are assigned by the tasks developers. To make the idea of weights clearer, suppose we have a symmetric encryption algorithm that uses three levels of security; level 1 that corresponds to a key length of 64 bits, level 2 of 128 bits and level 3 of 256 bits. Definitely using level 2 will give a strong encryption that the system can afford its computation overhead, therefore a weight of 1 can be given to this level while levels 1 and 2 can be given a less weight, e.g., 0.7, because level 1 has relatively weak key while level 2 has a very strong key that will cost more overhead with little improvement of encryption strength because level 2 is a strong level also.

To capture the importance of a security level k for a task i in comparison with other tasks in the system, a normalized level of the security is used, denoted as S_i^k, see Eq. (3.2).

$$S_i^k = \frac{k}{K_i} W_k^s, \quad k = 1, 2, \ldots, K_i \tag{3.2}$$

- **Integrated QoS and Security Metric (SQ).** In order to capture both QoS and security of a single admitted task we propose the SQ metric, given in Eq. (3.3), where a set of SQ levels are computed for each task. Each SQ level is the product of the normalized security level and normalized QoS level. The rationale behind using the product of both S and Q is that we aim to increase both to their highest possible values. Whereas using any other function like addition, weighted addition or division will not give the required result we are aiming to. The available CPU slack for task i to increase its QoS and

security levels is limited, let $slack_i$ denotes this slack. Our goal is to use this slack to have maximum almost equal S_i^k (normalized security level) and Q_i^l (normalized QoS level). The constraint $Exec\_S_i(k) + Exec\_Q_i(l) = slack_i$ should be satisfied. Considering addition of S and Q (S+Q) will not give a specific answer rather a set of answers satisfying the limited slack. Similarly the division of S by Q (S/Q) will only favor S over Q irrespective to the values of S and Q. See Fig. 3.2 which clarifies this argument for $slack = 8$, 10 QoS levels and 10 security levels. Computation overhead for QoS and security is equal to the level number, e.g. QoS level number 4 has an overhead of 4.

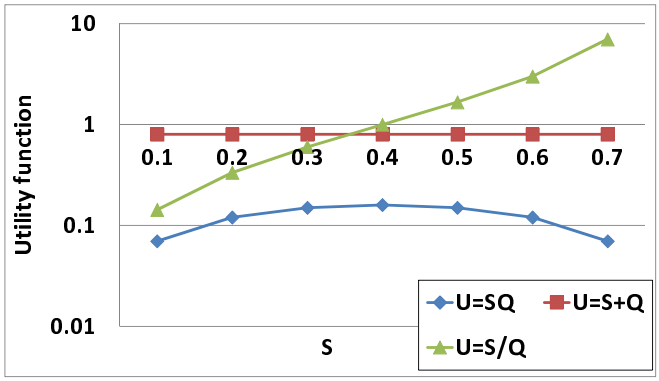


Figure 3.2 Utility function vs. S (normalized security) for one task

$$\mathbf{SQ_i} = [S_i^1 Q_i^1, \ldots, S_i^1 Q_i^{L_i}, \ldots, S_i^{K_i} Q_i^1, \ldots, S_i^{K_i} Q_i^{L_i}] \tag{3.3}$$

For SQ_i level b of security level k and QoS level l, the computation time ($Exec\_SQ_i(b)$) is $Exec\_S_i(k) + Exec\_Q_i(l)$. The CPU utilization by task i can be also represented as $U_i^b = (C_i + Exec\_SQ_i(b))/P_i$.

- **SQ to Utilization Ratio (SQUR).** This is the ratio of the first SQ level of task i to the utilization of the task with the first SQ level. The task with a highest SQUR (Eq. (3.4)) value is the task that most likely provides the system with higher SQ value. This parameter is used to guide the heuristic algorithm to choose the first task to increase

its SQ value.

$$SQUR_i = \frac{SQ_i^1}{U_i^1} \tag{3.4}$$

where SQ_i^1 is the first entry in SQ_i vector.

The goal is to choose the maximum SQ value for a given task that does not lead to a deadline miss for any task in the system. For the whole set of tasks in the system, the goal is to maximize the sum of all tasks current SQ (SQ_i^c) values which result in a new system wide measure of security-QoS value (denoted as SQV) as given in Eq. (3.5).

$$SQV = \sum_{\forall i \in \tau} SQ_i^c \tag{3.5}$$

- **System-wide Security Level (SSL).** SSL is the lowest security level among all tasks in the system. This parameter makes admission controller (to be discussed later) act fast when RL fluctuates. If SSL is greater than RL, then no need to take any action.

3.5.3 Problem Formulation

A set of tasks $\tau = \{T_1, T_2, \ldots, T_n\}$ has to be scheduled online on a single RTES, each task i has a set of instances $I(i)$ that are ready at the beginning of each period. Any task's instance will not relinquish the CPU before completion. Each task has its set of SQ values which are calculated for each task from its requested levels of security and QoS according to Eq. (3.3).

The goal of the system is to maximize the SQV value from running the accepted tasks. We refer to this problem as SQVMP (SQV Maximization Problem).

3.5.3.1 SQVMP

We now give the formal problem statement as: *Given a set* $\tau = \{T_1, T_2, ..., T_n\}$ *of tasks, each task is given as* $T_i = \langle C_i, R_i, Ql_i, Sl_i, D_i, P_i \rangle$*, find a feasible schedule that maximizes SQV of the set, while satisfying all the constraints (i.e. timing and security) in the schedule.*

Theorem 1. *SQV MAXIMIZATION PROBLEM (SQVMP) is NP-hard.*

Proof. We consider two special cases, by relaxing firstly the SQ levels, i.e. setting security and QoS levels to one for all tasks; and secondly the number of instances of the tasks to one and the arrival times of the tasks, i.e. setting $R_i = 0$ for all tasks i in the system, we refer to this problem as S-SQVMP, given in (3.6).

$$\textbf{Maximize } SQV = \sum_{i\in\tau}\sum_{k\in SQ_i} SQ_i^k X_i^k \tag{3.6}$$

subject to

$$\sum_{i\in\tau}\sum_{k\in SQ_i} U_i^k X_i^k \leq 1$$

$$X_i^k \in \{0,1\}, \qquad \forall i \in \tau, k \in SQ_i$$

First case is the problem of Dynamic Non-preemptive Real-Time Scheduling on a Single processor, which is proved to be NP-hard [47]. For the second case we need to state the Multiple Choice Knapsack Problem (MCKP) and then prove that our special case is a generalization to it.

MCKP can be stated as *" given a set k of item classes each has j items and each item has a profit p_i and a weight w_i, choose exactly one item j of each class N_i, i=1,...,k such that the sum of profits (p_i) is maximized without having the weight (w_i) sum to exceed the capacity of the knapsack c".*

$$\textbf{Maximize } \sum_{i=1}^{k}\sum_{j\in N_i} p_i^j x_i^j$$

subject to

$$\sum_{i=1}^{k}\sum_{j\in N_i} w_i^j x_i^j \leq c$$

$$\sum_{j\in N_i} x_i^j = 1 \qquad i = 1, ..., k$$

$$x_i^j \in \{0,1\} \qquad i = 1, ..., k, j \in N_i$$

The constraint from MCKP:

$$\sum_{i=1}^{k} \sum_{j \in N_i} w_i^j x_i^j \leq c$$

can be rewritten as:

$$\sum_{i=1}^{k} \sum_{j \in N_i} \frac{w_i^j}{c} x_i^j \leq 1$$

Then, S-SQVMP generalizes MCKP by mapping profits p_i to SQ_i, and w_i/c to U_i. The mapping can be done in polynomial time. Therefore a solution to S-SQVMP can be used to solve arbitrary instance of MCKP by doing the above mapping and presenting the tasks to a decision procedure of S-SQVMP.

The answer from S-SQVMP decision procedure is the answer to MCKP. Since MCKP is known to be NP-complete [53], S-SQVMP is NP-hard. Therefore the problem at hand (SQVMP) is NP-hard. □

3.5.3.2 MILP Formulation

Given a schedulable set of tasks, we can formulate the SQVM as a Mixed Integer Linear Program (MILP). A schedulable set of tasks composes a feasible schedule which is the assignment of tasks instances to be executed on the CPU without overlapping or missing deadlines. We denote the period $H = [0, \ P]$, where $P = \max\{R_i\} + 2LCM\{P_i\}, \ \forall i \in \tau$ and $LCM\{P_i\}$ is the least common multiple of all P_i's. We denote $Exec\_SQ_i(k)$ to be the execution time of SQ level k for task i. The scheduling should be performed over H to conclude that the task set is feasible [54]. $I(i)$ denotes the set of instances of task i. Each instance $j \in I(i)$ (starting from instance 0) of task $i \in \tau$ is started at S_{ij}, therefore the set $S = \{S_{ij}\}$ is a set of continuous variables that represents the resulted schedule. The binary variable X_{ijt} is one if and only if instance j of task i is started at time $t \in H$, otherwise it is zero. Binary variable X_i^k is one when SQ level k is chosen for task i, otherwise it is zero. We assumed that R_i, C_i, P_i, C_i^c and $Exec\_SQ_i(k)$ are integers, however if they are rational, we can multiply all of them by the LCM of their denominators.

$$\textbf{Maximize } SQV = \sum_{i \in \tau} \sum_{k \in SQ_i} SQ_i^k X_i^k \tag{3.7a}$$

subject to

$$C_i^c = C_i + \sum_{k \in SQ_i} Exec\_SQ_i(k) X_i^k, \qquad \forall i \in \tau \tag{3.7b}$$

$$\sum_{k \in SQ_i} X_i^k = 1, \qquad \forall i \in \tau \tag{3.7c}$$

$$\sum_{t=0}^{P - C_i^c} X_{ijt} = 1, \qquad \forall i \in \tau, j \in I(i) \tag{3.7d}$$

$$\sum_{i \in \tau} \sum_{j \in I(i)} \sum_{\acute{t}=max\{t - C_i^c + 1, 0\}}^{t} X_{ij\acute{t}} \leq 1, \qquad \forall t \in H \tag{3.7e}$$

$$S_{ij} = \sum_{t \in H} tXijt, \qquad \forall i \in \tau, j \in I(i) \tag{3.7f}$$

$$S_{ij} + C_i^c \leq S_{i[j+1]}, \qquad \forall i \in \tau, j \in I(i) \tag{3.7g}$$

$$S_{ij} + C_i^c \leq D_i(j), \qquad \forall i \in \tau, j \in I(i) \tag{3.7h}$$

$$S_{i0} \geq R_i, \qquad \forall i \in \tau \tag{3.7i}$$

$$X_{ijt} \in \{0, 1\}, \qquad \forall i \in \tau, j \in I(i), t \in H \tag{3.7j}$$

$$X_i^k \in \{0, 1\}, \qquad \forall i \in \tau, k \in SQ_i \tag{3.7k}$$

The objective (3.7a) follows from Eq. (3.5). Constraints (3.7b-3.7c) are satisfied when only one SQ level is chosen for task i. Constraint (3.7d) states that each instance starts and finishes within the interval [0 P], while constraint (3.7e) prohibits any simultaneous execution of different instances. Constraint (3.7f) relates S_{ij} to variable X_{ijt} and constraints (3.7g-3.7h) state that the instance j should finish before the start of next instance $j + 1$ and before the deadline where the deadline for instance j is $R_i + (j + 1)P_i$. Constraint (3.7i) is satisfied only if the first instance is started at or after its release time. Constraints (3.7j-3.7k) state that the variables used are only binary which can take either zero or one.

3.6 Security and QoS-aware Scheduling

In this section we propose the security and QoS aware scheduling algorithm. We first explain the system architecture and then discuss two scheduling task cases:

1. General SQV-aware scheduling where tasks are arrive dynamically to the system. The arrival times for the tasks in this case are arbitrary.

2. Mode change QoS-aware scheduling, where the tasks are assumed to be arrived at the same time with one instance. This case is a relaxed version of the previous case.

3.6.1 System Architecture

The block diagram of the SQV-aware scheduling system is shown in Fig. 3.3. The scheduler system is running on a single RTES node. This system consists of three stages; the admission controller, the SQV optimizer and the ordinary EDF scheduler.

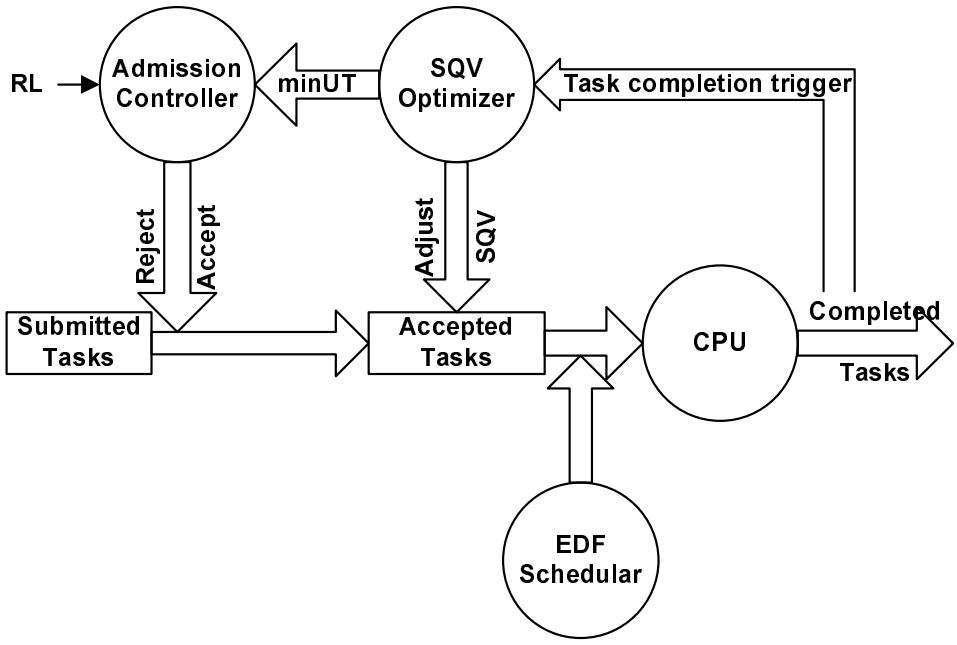


Figure 3.3 SQV-aware scheduling system

Fig. 3.4 shows the admission controller pseudo code. The admission controller checks every arriving task against the system parameters (utilization bound, security level and deadline miss

of any task in the system). Performing the feasibility test of dynamically non-preemptable tasks is an NP-hard problem [47]. Therefore we use a sufficient feasibility test proposed in [55]. This test is:

$$\sum_{\forall i \in \tau} U_i \leq 1 - \max_{k \in \tau} C_k^c \left(\frac{1}{P_s} - \frac{1}{P_k}\right) \tag{3.8}$$

where P_s is the shortest period in the task set.

Input: Arriving tasks.
Output: Admitted tasks parameters (SQUR, U and SQ) and minimum utilization ($minUT$).

1: **for all** tasks $i \in arriving$ **do**
2: Calculate i's minimum utilization U_i.
3: Calculate $SQUR_i$ (Eq. (3.4)).
4: Calculate SQ_i set (Eq. (3.3)).
5: Sort SQ_i in increasing order of its values.
6: **end for**
7: $ArrivedSorted \leftarrow$ tasks in increasing order of SQUR.
8: **for all** $i \in ArrivedSorted$ **do**
9: **if** (Eq. (3.8)) is satisfied and $K_i \geq RL$ **then**
10: Admit i.
11: Update min. utilization ($minUT$) of the system.
12: **else**
13: Reject task i
14: **end if**
15: **end for**

Figure 3.4 Admission controller algorithm

Fig. 3.5 shows the SQV optimizer algorithm. The SQV optimizer optimizes the SQV of the admitted tasks. In order to respond to the system dynamics, i.e., task arrival or departures; or risk level fluctuations, the SQV optimizer is triggered. The SQV optimizer modifies (increase or decrease) the integrated security and QoS levels (combined in the SQ levels) so as to respond to system dynamics. Then EDF scheduler chooses tasks for running on the node CPU. The system works in steps as shown in 3.3.

- **Step 1:** The admission controller (AC) called upon arriving of tasks or fluctuation of risk levels. AC checks for several parameters like the system security level (SSL) and

Input: All tasks in the system and system parameters (RL, SSL, minUT).
Output: SQV, SSL and tasks update of SQ levels.

1: **if** $RL > SSL$ OR a task arrives or departs **then**
2: $Sorted \leftarrow$ tasks sorted in decreasing order of SQUR.
3: $Max = MaxTMP \leftarrow \max_{\forall i \in \tau} C_i(\frac{1}{P_s} - \frac{1}{P_i})$
4: **for all** $i \in Sorted$ **do**
5: $UT \leftarrow minUT - U_i$
6: Increase SQ_i level for task i to maximum possible value
7: Update U_i with current SQ_i reflected in C_i^c
8: $MaxTMP \leftarrow \max\{Max, C_i^c(\frac{1}{P_s} - \frac{1}{P_i})\}$
9: **while** $UT + U_i > 1 - MaxTMP$ **do**
10: Decrease SQ_i for task i
11: Update $MaxTMP$ and U_i
12: **end while**
13: $Max \leftarrow \max\{Max, MaxTMP\}$
14: $UT \leftarrow UT + U_i$
15: Update SQV and SSL.
16: **end for**
17: **end if**

Figure 3.5 SQV optimizer algorithm

available minimum utilization (Sum of task utilizations where tasks in the system are in the lowest SQ level). AC uses Eq. (3.8) to decide if a task can be accepted. Then, the task is accepted if it passes the test, otherwise it is rejected. In this stage, SQ levels are also calculated for the arrived tasks and sorted in a non-decreasing order of their values. If two SQ values are equal, the one with the smaller computation time is chosen (ties are broken arbitrarily).

- **Step 2:** The SQV optimizer optimizes the SQ levels for the tasks in the system taking into account the timing and security constraints of all accepted tasks.

- **Step 3:** The EDF scheduler runs on the optimized tasks.

The above steps are repeated each time a new task arrives, a task finishes, or a risk level changes, while the system parameters are logged. When there is any change in the risk level reported by the STE, the system will compare the risk level with the SSL. If the risk level is

higher than SSL, all the tasks in the system are marked as arriving tasks and the steps above are repeated. If the running task has a lower security level than the risk level, this task can be deleted.

3.6.2 General SQV-aware Scheduling Algorithm

Fig. 3.6 shows the flowchart of the optimizer algorithm. The SQV_EDF algorithm works as a central part of the optimizing system, on dynamically arriving tasks to the system. The tasks are sorted in a decreasing order of SQUR, Eq. (3.4). If SQUR ratio for a task is high, the participation in the overall SQV would be higher, while the computation demand is lower. SQUR is different for various SQ levels of the task, but we consider the first SQ level for the sake of simplifying the computations. The SQV optimizer picks the task with the highest SQUR value. Then, starting by its maximum SQ level, SQV optimizer checks for any deadline miss. If there is a deadline miss, the optimizer decreases the SQ level until the schedule is feasible. This step is repeated until there is no utilization slack left or there are no more tasks to be optimized.

3.6.3 Mode Change QoS-Aware Scheduling

When the mode (e.g., the UAV modes are; takeoff, normal cruise and landing) of operation for the node changes, a new set of tasks should be admitted [56]. This set of tasks is actually the same set in the model we used so far, but the arriving times are relaxed to be zero and only first instance of each task is considered. Therefore the set of tasks is considered as a non-periodic and preemptable. For this case we use the same system but the feasibility check now is simpler, we need to check that the sum of tasks utilizations are not more than one (EDF utilization bound) as proved by Dhall and Liu in [57]. We assumed all the tasks from the previous mode are completed and we need to schedule a new set. In fact this problem can be formulated as in 3.6

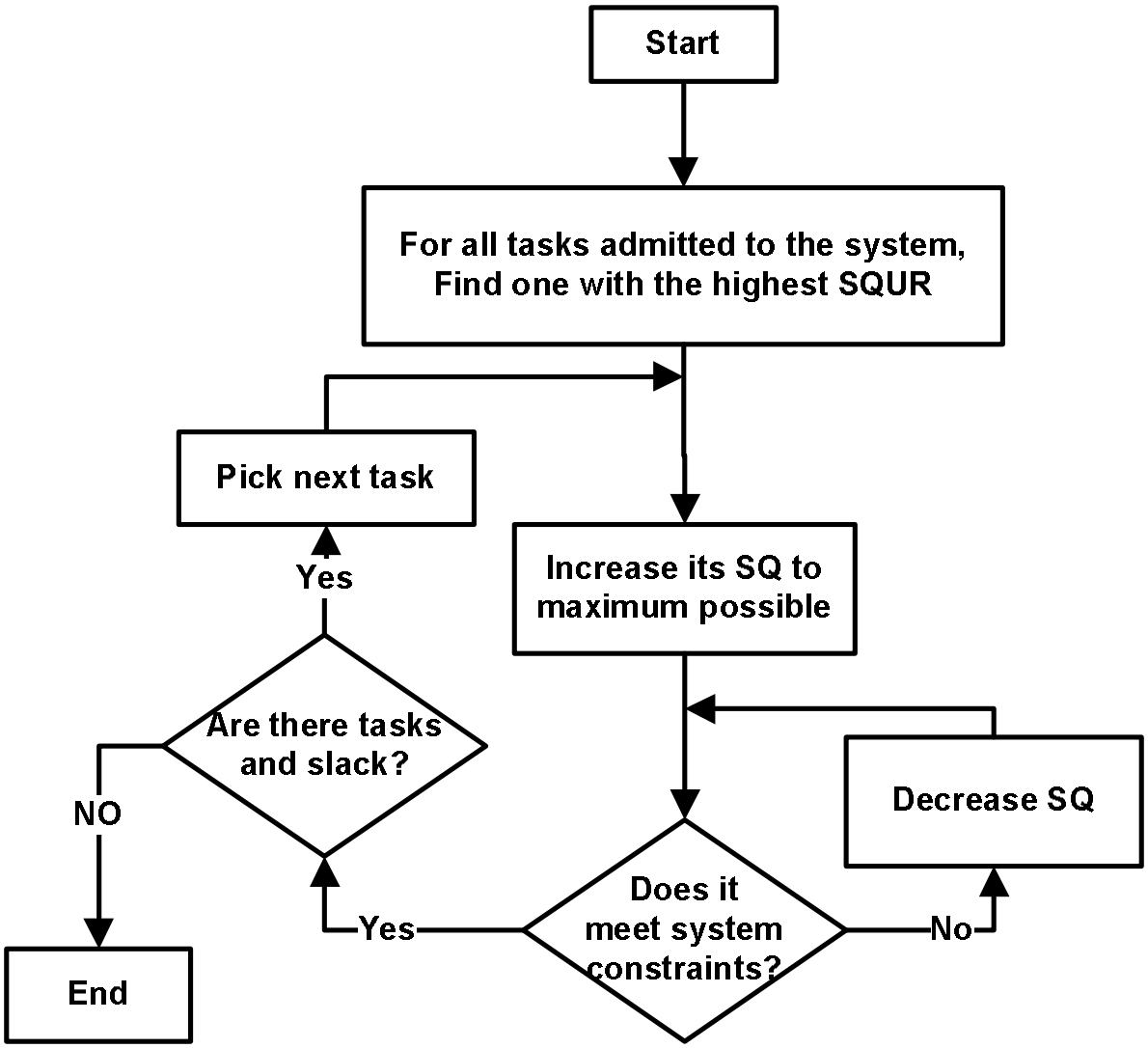


Figure 3.6 Flowchart of SQV-optimizer

3.6.4 Algorithm Complexity

Now, we evaluate the time complexity of our algorithm.

Theorem 2. *The time complexity of the scheduling algorithm SQV_EDF is* $O(n(n+qslog(qs)))$*, where* n *is the number of tasks,* q *is the greatest number of QoS levels and* s *is the greatest number of security levels between the tasks.*

Proof. The time complexity of the algorithm can be analyzed by evaluating the time complexity of the steps involved in the algorithm. Step 2 and 3 in Fig. 3.4 take time $O(1)$, step 4 takes time

Table 3.1 Example task set

T_i	C_i	R_i	Ql_i	Sl_i	P_i
$T1$	1	0	{0, 1, 2, 3, 4}	{1, 2, 3}	10
T2	1.5	0	{0, 1.5, 3, 4.5, 6, 7.5}	{1.5, 3}	15
T3	3	0	{0, 3, 6, 9, 12, 15}	{3, 6, 9}	30
T4	1.5	0	{0, 1.5, 3}	{1.5, 3, 4.5}	45

$O(qs)$ and step 5 takes time $O(qslog(qs))$. Hence the for-loop, 1-6, for n tasks in the system takes time $O(n(qs+qslog(qs)))$. Step 7 takes time $O(nlogn)$. Step 9 takes time $O(n)$, hence the for-loop, 8-15, takes time $O(n^2)$. Therefore the admission algorithm takes $O(n(n+qslog(qs)))$

In Fig. 3.5, step 2 takes time $O(nlogn)$, step 3 takes time $O(n)$ to find the maximum. Steps 5-9 each takes time $O(1)$. Steps 11-12 each takes time $O(1)$, therefore while loop, 10-13, takes time $O(qs)$. Steps 14-16 each takes $O(1)$. For loop 4-17 takes $O(nqs)$ to optimize all tasks in system. The time complexity of optimizer algorithm, Fig. 3.5, is $O(nlogn + nqs)$.

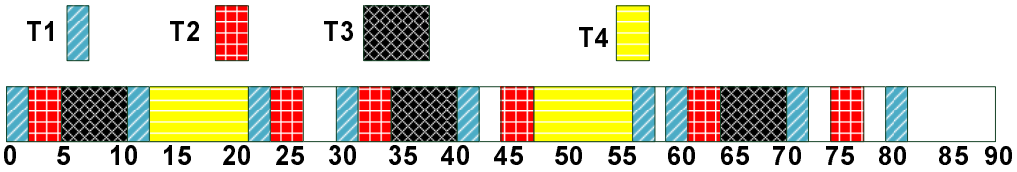
Therefore the time complexity of the SQV_EDF is $O(n(n + qslog(qs)) + nlog(n) + nqs) = O(n(n + qslog(qs)))$ □

3.7 An Illustrative Example

In order to illustrate the operation of the proposed algorithm, consider the set of tasks given in Table 3.1.

All the tasks arrive at time 0 and all the weights are set to 1 with the parameters shown in the figure. Further, for simplicity, we assume the tasks are preemptable in order to use the feasibility check of Liu in [57], i.e., $\sum U_i \leq 1 \ \forall i \in \tau$. We need to schedule those tasks on a single node to maximize the resulting SQV while meeting the deadlines of the maximum number of tasks. To simplify the computation we assume that the risk level is 1.

The first step is to run the admission controller on the arrived tasks. All the tasks are admitted since there maximum security levels are greater than the risk level and the sum of their utilizations is less than 1. Fig. 3.7(a) shows the schedule, up to hyper period (i.e., the LCM of the tasks periods), for the tasks with their first QoS and security levels, using EDF scheduling algorithm. In this step, SQURs and SQ sets are calculated for all the admitted tasks



(a) Schedule without optimization

T1 (Q: 1 2 3 4 5)

S	1	2	3	4	5
1	0.07	0.13	0.2	0.27	0.33
	(1)	(2)	(3)	(4)	(5)
2	0.13	0.27	0.4	0.53	0.67
	(2)	(3)	(4)	(5)	(6)
3	0.2	0.4	0.6	0.8	1
	(3)	(4)	(5)	(6)	(7)

SQ/U=0.07/0.2=0.35

T2 (Q: 1 2 3 4 5 6)

S	1	2	3	4	5	6
1	0.06	0.11	0.17	0.22	0.28	0.33
	(1.5)	(3)	(4.5)	(6)	(7.5)	(9)
2	0.11	0.22	0.33	0.44	0.56	0.67
	(3)	(4.5)	(6)	(7.5)	(9)	(10.5)
3	0.17	0.33	0.5	0.67	0.83	1
	(4.5)	(6)	(7.5)	(8.5)	(10.5)	(12)

SQ/U=0.06/0.2=0.3

T3 (Q: 1 2 3 4 5 6)

S	1	2	3	4	5	6
1	0.06	0.11	0.17	0.22	0.28	0.33
	(3)	(6)	(9)	(12)	(15)	(18)
2	0.11	0.22	0.33	0.44	0.56	0.67
	(6)	(9)	(12)	(15)	(18)	(21)
3	0.17	0.33	0.5	0.67	0.83	1
	(9)	(15)	(18)	(21)	(24)	(27)

SQ/U=0.06/0.2=0.3

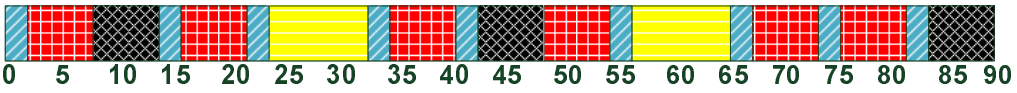
. SQ value
(.) Execution time

T4 (Q: 1 2 3)

S	1	2	3
1	0.11	0.22	0.33
	(1.5)	(3)	(4.5)
2	0.22	0.44	0.67
	(3)	(4.5)	(6)
3	0.33	0.67	1
	(4.5)	(6)	(7.5)

SQ/U=0.11/0.07=1.65

(b) SQUR and SQ sets



(c) Final schedule

Figure 3.7 The illustrative example of the SQV-optimizer algorithm

as in Fig. 3.7(b), where the values between parentheses under SQ values are the execution times of SQ levels. The SQ set is sorted in non decreasing order and the execution times of SQ levels are ordered accordingly.

The resulting schedule has a utilization slack of 1-0.67=0.33. This slack can be utilized by increasing the SQ level of some tasks. The task that will get this opportunity is the task with the greatest SQUR value. Task 4 has the greatest SQUR value ($= RL/$(No. of Security levels)$\times$ $1/$(No. of QoS levels)$\times P_4/(C_4 + Exec\_SQ_4(1)) = 1.65$). Therefore we start with task 4's maximum SQ value and check for schedulability of all tasks. All tasks are schedulable, because it is sufficient in this case to check the utilization sum, which is 0.8 <1 and the resultant SQ is 1. We still have a slack of 1-0.8=0.2 in the schedule. Next task is task 2 its SQUR is 0.4, by doing the same for task 4 we stopped at SQ=0.33 (QoS level=2 and security level=2). After that there is no slack and the algorithm stops.

The SQV for the whole set of tasks is $SQ_1^1 + SQ_2^5 + SQ_3^1 + SQ_4^9 = 0.07 + 0.33 + 0.06 + 1 = 1.46$. The final schedule is given in Fig. 3.7(c).

3.8 Simulation Studies

In this section, we describe the simulation setup to evaluate the proposed algorithm. For the purpose of simulations, we use the same task parameters (see Table 3.2) as in [17]. We add to each task a set of security levels and a set of QoS levels in the range [1,2,...,10].

We use three baseline algorithms for the sake of comparison with our algorithm, which are the same used in [30] with a little modification to suite our simulations:

- MAX_EDF: Uses the maximum SQ value of the task to be run. It admits tasks according to highest SQ value.
- MIN_EDF: Uses the minimum SQ value for the task during admittance and running in the system.
- RND_EDF: Randomly chooses the SQ value for the task and then try to admit the task.

Only Q values are considered when the above baseline algorithms are compared with QoS_EDF.

All of the baseline algorithms use the feasibility check in Eq. (3.8). All n tasks need to be checked for feasibility using Eq. (3.8) and each task needs a calculation and sorting of its SQ values, which takes time $O(qslog(qs))$ in the worst case. Therefore the time complexity for each one of the baseline algorithms is $O(n + qslog(qs))$.

Table 3.2 Task parameters

Exec. time (C_i)(ms)	U[3 8], uniformly distributed
QoS levels	U[1 10], uniformly distributed
Security levels	U[1 10], uniformly distributed
Deadline D_i(ms)	U[8 12]*C_i

We assume throughout our simulation studies, the message computation time for the first security level is equal to C_i. In the absence of real published data, we use a linear overhead models for security ($Exec\_S_i(k) = M_i(k/K_i)$, where M_i is the message computation time with first security level for task i and k is the security level) to calculate the computation times for security levels and for QoS levels computation times ($Exec\_Q_i(l) = C_i(l/L_i)$, where; l is the QoS level). We assume further that the weights of the QoS and security levels, W^q and W^s, for all levels are 1, for the sake of simple calculations. We define three parameters to be used in the performance comparison: Success Ratio (SR), Dropping Ratio (DR) and Admittance Ratio (AR) as follows:

$$SR = \frac{\text{No. of completed tasks}}{\text{No. of arrived tasks}} \tag{3.9}$$

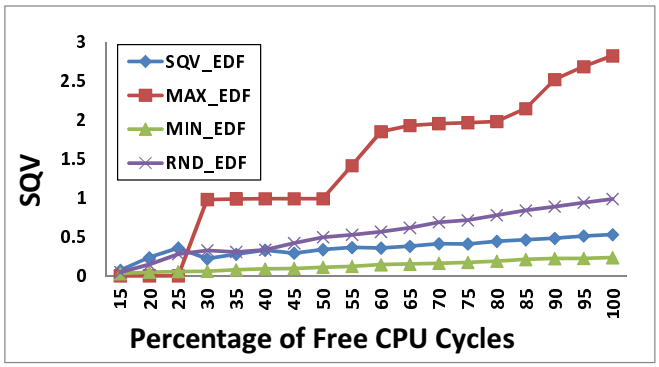
$$DR = \frac{\text{No. of dropped tasks}}{\text{No. of admitted tasks}} \tag{3.10}$$

$$AR = \frac{\text{No. of admitted tasks}}{\text{No. of arrived tasks}} \tag{3.11}$$

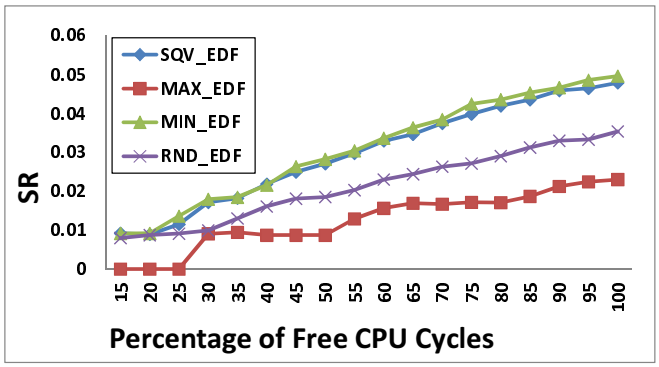
Note that it is not necessary that all admitted tasks will complete their execution. Some of them will be dropped if the system status varied, i.e. if the task maximum security level is lower than the new risk level it will be dropped.

3.8.1 Simulation Model

In order to evaluate the proposed algorithm, four simulation studies were conducted;



(a) SQV vs. CPU utilization



(b) SR vs. CPU utilization

Figure 3.8 SQV and SR vs. CPU utilization

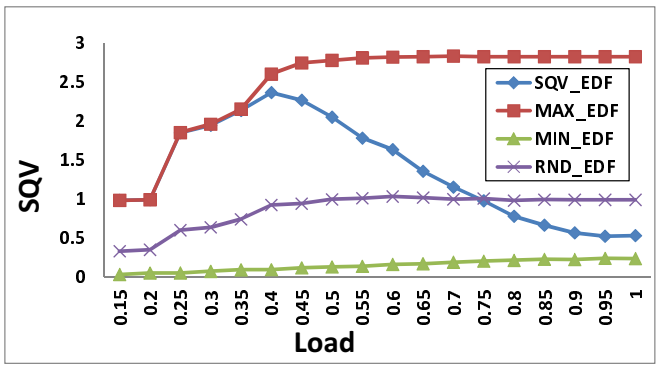
(i) The percentage of free CPU cycles (CPU utilization) (assuming the worst case execution time (WCET) of the task) is variable and the risk level is kept at level one. This part demonstrates when a node has other tasks running and there is only a fraction of the CPU cycles available for the arrived tasks. We generate enough tasks to use up the available CPU cycles.

(ii) The CPU is free and the risk level is at level one. We vary the sum of the arrived task utilizations over the range [15%,. . . ,100%].

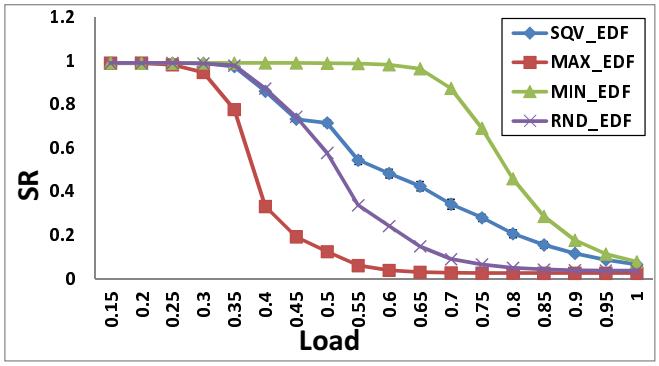
(iii) The risk level is dynamically varied at a rate that is about five times the average life time of the tasks in the system. This gives enough time for arrival and departure of the tasks. SQV, DR and AR are logged along the time line of the simulation. The CPU is free. The task parameters in Table 3.2 are kept the same in all simulation studies.

(iv) Mode change QoS-aware scheduling. The set of task parameters are same as shown

in Table 3.2, except that the tasks are assumed to be ready at time 0 and their periods are increased to provide an average utilization of 0.01. The generated tasks are scheduled using the proposed algorithm. The same tasks are formulated as an ILP instance Eq. (3.6) and given to CPLEX program [58]. Then the results are compared with SQV-algorithm.



(a) SQV vs. Load



(b) SR vs. Load

Figure 3.9 SQV and SR vs. Load

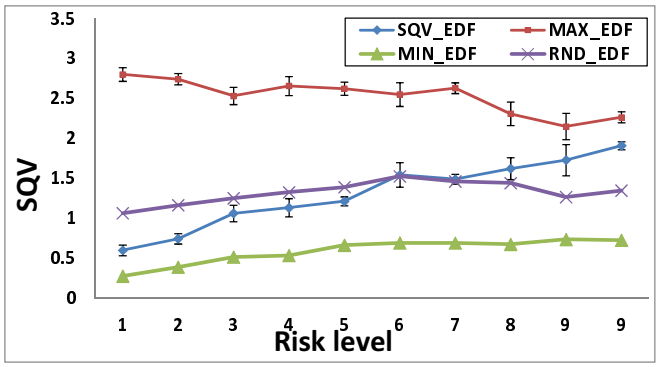
We ran the three baseline algorithms for the first three parts to compare with SQV_EDF. We used the same system (i.e., SQV-aware scheduling system) to run the baseline algorithms, after switching off the SQV_optimizer. For each point in the figures we ran the simulation 15 times with IID (Independent and Identically Distributed) tasks arrived with exponentially distributed inter arrival times and then we took the average. The maximum 95% confidence intervals are very small to be plotted for the figures of part one and part two of our simulation studies. For the third part, the 95% confidence intervals are shown in the figures.

For all the simulations, we used a suitable arrival rate such that we always have enough

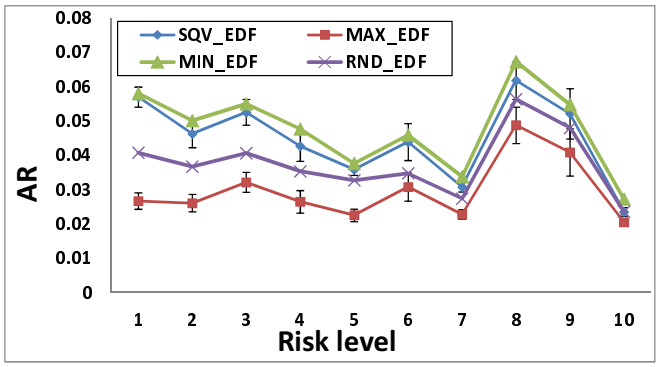
arriving tasks to the system. After running some instances (the number of instances is drawn from an exponential departure process) the task will depart the system as a completed task.

3.8.2 Results and Analysis

The goal of the SQV-aware scheduling algorithm is to maximize the number of admitted tasks and to maximize the SQV of the admitted tasks. There is a tradeoff between these two requirements. The SQV_EDF algorithm tries to admit more tasks then works on the admitted tasks to raise the SQV of the system. Therefore the performance in the simulations is close to SQV_MIN in terms of SR.



(a) SQV vs. Risk Level



(b) AR vs. Risk Level

Figure 3.10 SQV and AR vs. increasing Risk Level

3.8.2.1 Effect of CPU Utilization

Fig. 3.8 shows the results of the first simulation study, where the percentage of free CPU cycles is varied from 15% to 100%. Fig. 3.8(a) shows that MAX_EDF has a higher SQV than other algorithms. Fig. 3.8(b) shows that SQV_EDF outperforms MAX_EDF and close to MIN_EDF in terms of success ratio (SR). This is a good performance in the sense that SQV_EDF admits more tasks than MAX_EDF and RND_EDF and optimizes their QoS and security levels to values greater than MIN_EDF. MAX_EDF tends to admit tasks according to their highest SQ values; therefore the resulted SQV for small number of admitted tasks is high. SQV_EDF benefits from any utilization slack and tries to elevate system's SQV (QoS and security levels), which makes its performance, in terms of SQV, better than MIN_EDF. In all simulation studies, MIN_EDF is outperformed, in terms of SQV, by all other algorithms.

MIN_EDF accepts as many tasks as possible without taking into account any improvement in security or QoS. This algorithm uses the lowest SQ value, which makes this algorithm unable to use up the available slack. This results in more admitted tasks, but in a low SQV value.

On the other hand, RND_EDF admits tasks on a random basis which results in a good performance but without any guarantee that the chosen SQ level will admit the task, which consequently results in a bad SR (see Fig. 3.8(b)).

3.8.2.2 Effect of Tasks Load

Fig. 3.9 shows the effect of tasks load on SQV and SR of the system. The load is varied in terms of the minimum utilization of the tasks while the system is free. Therefore the admitted tasks can utilize the whole CPU available cycles.

Up to about 40% of the load in Fig. 3.9(a), the SQV_EDF algorithm performance is close to MAX_EDF. After that, SQV_EDF performance declined and MAX_EDF stayed at the same level. Again, the SQV_EDF performance cannot be taken without taking SR into account, where its performance is better than MAX_EDF and RND_EDF as shown in Fig. 3.9(b).

MAX_EDF algorithm keeps increasing the number of admitted tasks up to 50% of the load. Then the number of admitted tasks becomes fixed while the number of arrived tasks increases,

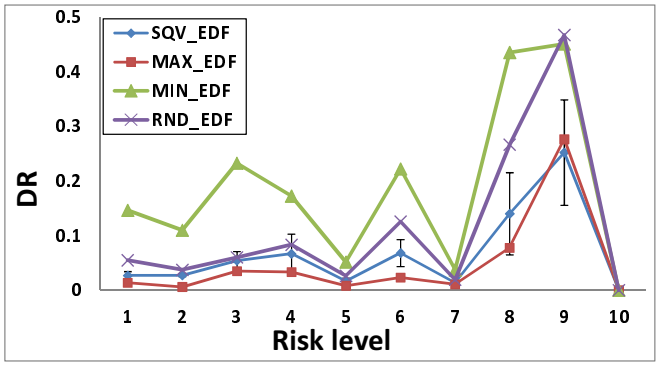


Figure 3.11 DR vs. increasing Risk Level

which give a saturated SQV values up to the end of 100% of the load and a bad performance in terms of SR.

SQV_EDF algorithm has the ability to modify the QoS and the security levels of the admitted tasks. As the load increases, the goal of SQV_EDF algorithm becomes admitting as many tasks as possible. Therefore, the SQV is lower than MAX_EDF and at later point on the curve, is also lower than RND_EDF. Both MAX_EDF and RND_EDF have a lower success ratios at those points as shown in Fig. 3.9(b).

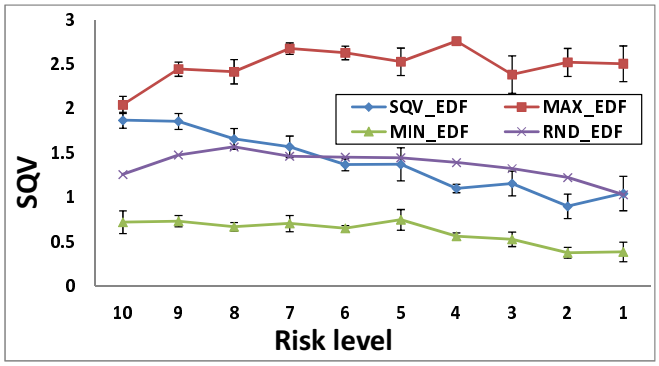
3.8.2.3 Effect of Risk Level

Fig. 3.10(a) and 3.12(a) show the SQV vs. a varied number of risk levels. SQV_EDF performance is less than MAX_EDF due to the greater number of tasks admitted by SQV_EDF and better than MIN_EDF, due to the ability to optimize the admitted tasks.

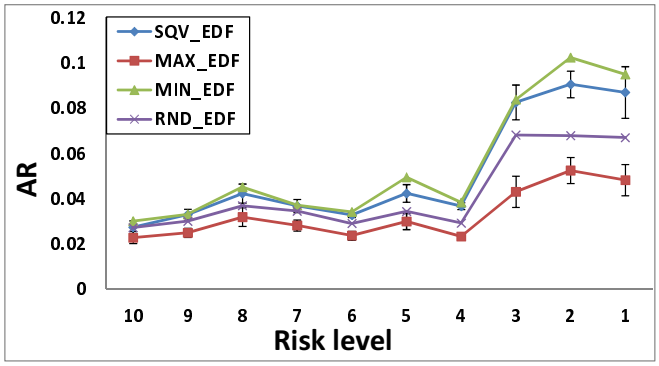
Fig. 3.11 shows the dropping ratio of the admitted tasks vs. the risk level. MIN_EDF has the highest dropping ratio while the other algorithms have almost the same dropping ratios. This dropping behavior of tasks is expected because of the increase in risk level which makes some tasks unable to support the lowest required security level (i.e. max. security level of the task is less than the risk level).

When the risk level decreases, see Fig. 3.12(b), no task will be dropped because all admitted tasks have the required minimum security level. The admitted to arrived ratio for SQV_EDF is close to MIN_EDF and higher than the others. This reflects the power of adaptability of the

SQV_EDF algorithm to the variation in the risk level.



(a) SQV vs. Risk Level



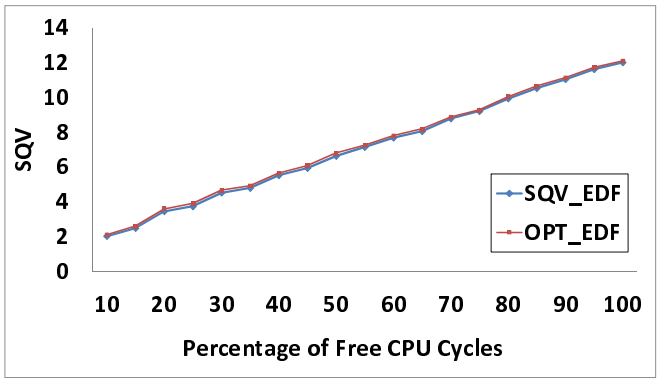
(b) AR vs. Risk Level

Figure 3.12 SQV and AR vs. decreasing Risk Level

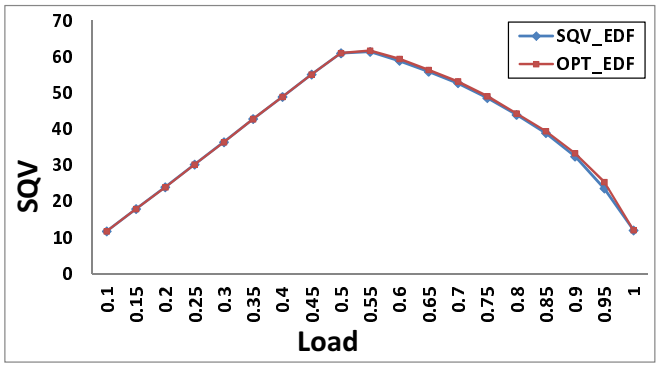
3.8.2.4 Mode Change Scheduling

SQV_EDF algorithm tries to raise the SQV value of the system to the maximum possible level. To show this tendency, simulation part (iv) is conducted and Fig. 3.13(a) and 3.13(b) show the results.

Fig. 3.13(a) shows that when percentage of the available CPU cycles is low, no algorithm can optimize the admitted tasks because of the lowest available slack. As the slack increases the algorithms (SQV_EDF and optimal) have the ability to raise the SQ values of the tasks. SQV_EDF raises the SQ values using a heuristic measure (SQUR) while the optimal tries all possible solutions and picks the solution with the largest SQV. The SQV_EDF is very close to optimal as shown in the figure.



(a) SQV vs. CPU utilization



(b) SQV vs. Load

Figure 3.13 SQV vs. Load and CPU utilization

In Fig. 3.13(b), SQV_EDF and the optimal solutions are the same up to 60% of the load where the slack is enough to raise SQ values of the tasks to the highest levels. As the number of arrived tasks increases, the slack becomes lower and SQV_EDF tends to optimize the admitted tasks which gives close performance to optimal up to 100% of the load where neither SQV_EDF nor optimal has the ability to increase the SQ values because of the lack in CPU free cycles.

In summary, the SQV_EDF algorithm performance is very close to the upper bound SQV and has a SQV value which is less than MAX_EDF and RND_EDF under overload conditions. However it has higher success ratio as opposed to MAX_EDF and RND_EDF which is close to MIN_EDF. This makes the overall performance of SQV_EDF better than other baseline algorithms in all the simulations and close to the optimal.

3.9 Conclusions

In this chapter, the problem of maximizing security and QoS of the tasks admitted to an RTES node while maintaining the timing constraint of these tasks is considered. The problem is formulated as an optimizing problem of the combined security and QoS of the system (SQV). A new metric (SQV) that jointly considers security and QoS is identified and used as the optimization metric in the scheduling optimization problem. Then a heuristic algorithm (SQV_EDF) to obtain a polynomial time solution for this optimization problem is developed. The SQV_EDF algorithm works by admitting tasks according to their minimum utilization. Then, using the available slack, it increases SQ level of the most beneficial task that adds to SQV more than other tasks in the system. In order to evaluate the proposed algorithm, a wide range of workloads and several system conditions are considered in the simulation studies.

The proposed algorithm outperforms a number of baseline algorithms in achieving a better performance of the system considering the SQV and SR of the tasks available to the system. The algorithm performs very close to MIN_EDF algorithm in terms of success ratio which makes this algorithm appropriate to be used in critical systems where the security and QoS characteristics of the task permit for optimization.

CHAPTER 4 Static DAG Allocation in Distributed Real-Time Systems

4.1 Summary

Heterogeneous distributed real-time systems are continuously evolving to realize many emerging mission critical applications, e.g., battle field vision systems. In such systems, there often exists a tradeoff between quality of results and security of task execution while satisfying real-time constraints. In this chapter, we consider a set of dependent real-time tasks, modeled as Directed Acyclic Graph (DAG), with security and QoS requirements for static assignment and scheduling on a set of heterogeneous sites with the objective of maximizing Total Quality Value (TQV) of the system. This problem is NP-hard since the basic problem of scheduling a DAG on multiple processors is NP-hard. We make the following contributions; (i) define new metric, TQV, which captures QoS aspects of the DAG and helps in choosing a task in the task graph, DAG, to increase its QoS level so as to raise system TQV to the best value, (ii) based on the defined metric, we propose a polynomial time heuristic algorithm to maximize TQV, and (iii) we evaluate the algorithm through simulation studies by comparing it to baseline algorithms for variations of synthetic workloads. The proposed algorithm outperforms the baseline algorithms in all the simulated conditions for fully-connected and shared bus network topologies.

4.2 Background

Distributed real-time systems have become increasingly evolved in many aspects of our life. Such systems are widely used in industry and military applications, e.g. factory automation,

smart power grid and battle field vision systems [15, 16]. Real-time system is that system where the correctness of the system does not depend only on the logical results of the computations, but also on the time at which the results are produced [59]. Further when this system is deployed in an open (public) environment, the exchanged data should be secured against any modifications, replications or even eavesdropping by unintended recipients. Security infrastructure is assumed to be there to estimate the risk level of using the underlying network. Therefore real time tasks should meet the security (at least match the risk level) and deadline constraints. When a deadline is missed the result can be useless or even catastrophic consequences may happen, e.g. in missile tracking systems.

Consequently, to guarantee that every real-time task will produce acceptable quality results while meeting the constraints of timing imposed by the system and security imposed by the environment, effective allocation algorithms should be employed in distributed real-time systems. The allocation means the assignment of tasks onto processors and then scheduling tasks on the processors to achieve the system goal.

Based on the idea of imprecise computation introduced by Lin et al. [21], a real time task consists of two parts: Mandatory part and optional part. If the task has time to complete both parts the result is said to be precise; whereas the task result is said to be imprecise, if not all parts are fully completed. Hence the optional part refines the result of the mandatory part. If the optional part is divided into several portions the level of refinement and hence the level of accuracy is proportional to the number of completed optional portions.

As an example, for a team of Unmanned Aerial Vehicles (UAVs), in Fig. 2.3, which are cooperating to monitor a battle field to localize, detect or recognize specific targets, the tasks can be allocated to those nodes (UAVs) either before the beginning of the flight or during the flight when the mission is changed. In this case, the task model of imprecise computation allows some tasks to run in a lower accuracy level, so that the result accuracy of the mission is acceptable while timing and security constraints are satisfied.

In distributed real-time systems, applications usually consist of several tasks, where the output of a task is used as an input to another task. This imposes precedence constraints

among the tasks in a form of Directed Acyclic Graphs (DAGs) that have an end-to-end deadline. Tasks in the DAG do not have any individual deadlines rather an end-to-end deadline over all the tasks in the application that must be met. The task cannot start execution unless it received all inputs from its immediate precedent tasks in the graph.

Each task in the DAG is assumed to have several optional portions and a mandatory part. For a task to produce an acceptable accuracy, at least its mandatory part must be completed. The optional portions refine the result produced by the mandatory part. That is, the accuracy (QoS) of the result is further increased, if more optional portions of the task are allowed to be executed. Partially accurate (imprecise) input affects the output of the task and the output tends to be partially accurate too, even if the task has time to execute all of its optional parts. More specifically, to produce a high quality result of the application (produced by the cooperation of all tasks), the available processing time should be effectively allotted between tasks. An application is considered to be feasible if all component tasks have at least completed execution of their mandatory parts before the application's deadline.

In this chapter, we address the problem of allocation of precedence constrained-tasks to heterogeneous sites in a distributed security-sensitive system. We consider two types of communication network topologies:

(i) Fully connected − each node has a contention-free communication channel to all other nodes;

(ii) Shared bus − all nodes share the same channel and contend for using it.

The goal is to find a feasible allocation of tasks to sites that maximizes the objective value (i.e., TQV, to be described later). By feasible allocation, we mean to find a feasible schedule under a given assignment that meets the constraints of timing, precedence and security; and maximizes the objective value. The main characteristics of the problem are:

C1. Tasks within an application are communicating with each other during the course of the mission to achieve the system goal, which in turn forces the precedence relationship among communicating tasks. The precedence constraints have to be accounted for during the allocation of tasks to sites.

C2. Tasks have the ability to run in several QoS levels according to available execution time on the processor and their available time to deadline.

C3. System deployment area imposes the security constraint where information exchanged among tasks via the network should be secured in a level that is equal to a pre-estimated threat level of the deployment area.

C4. The workload (application) has to finish before its deadline with acceptable performance level.

C5. Sites that are available for executing the given applications are heterogeneous in terms of processing capabilities. Throughout this chapter we use processor and site interchangeably.

C1, C3, C4 and C5 give the characteristics of the task and system models under consideration, while C2 provides the direction of the objective function. C2 describes tasks abilities to run in a range of performance levels according to available running time [52], which results in an overall application performance that is dependent on how the available time is distributed between the tasks. Some tasks can improve the application performance better than others if they have used the same available time, this difference in quality improvements is merely related to the nature of the processing done by the task and it is beyond the scope of this dissertation. We call the application performance suggested by C2; Total Quality Value (TQV), then we aim to maximize TQV by carefully choosing which task will participate more to it, to raise its QoS level.

The rest of this chapter is organized as follows. The related literature is reviewed in Section 4.3. Problem statement and the system model are given in Section 4.4. In Section 4.5 the QoS-aware allocation algorithm is discussed. Simulation studies are shown and discussed in Section 4.6. In Section 4.7 Branch and Bound heuristic algorithm is described. Finally, Section 4.8 concludes the chapter.

4.3 Related Work

Many researchers have considered security and QoS requirements when studying resource management in time critical systems. However they either studied one requirement (e.g., [17,19,30,31] studied security dimension and [32,39,60–62] studied QoS dimension), or restrict their study on single processor system and assuming the tasks are preemptable [33]. For those who studied QoS and security in distributed systems they assumed the task set is independent [18,63].

Allocation of applications, modeled as DAGs, on a set of distributed processors has been extensively studied by the community. Two general methods are used to assign DAGs to distributed processors. First method is the clustering based, where tasks are clustered according to some criteria, e.g. in [38] tasks are clustered according to the ratio of the communication cost to the computation cost of the communicating pairs of tasks. Then assignment to actual sites is taken place. Second method is the list based assignment where tasks are given priorities according to its importance in the graph. The task with highest priority is considered for assignment first. After assignment the scheduling stage is taken place in both assignments. Non-preemptive scheduling is preferred over preemptive scheduling in many real time systems for its lower overhead and ability to prevent deadlock [20]. The survey [41] gives a detailed discussion of DAGs assignment and scheduling on distributed systems.

In [39] the authors used a list assignment policy. They addressed the effect of error in input to a task in an application modeled as a DAG after partially completed preceded tasks. They proposed a modified versions of well known algorithms (i.e., EDF, LSTF, and HLF), to dynamically allocate tasks on a homogeneous distributed real-time system. In this chapter, we use EDF (EDDF) list assignment policy. We assume in our work the processors (sites) are heterogeneous and we consider the security constraint of the underlying network.

The problem of off-line allocation of a set of periodic Directed Acyclic Graphs (DAGs) on a set of heterogeneous sites (processors) is considered. In this chapter, the QoS and security requirements of the nodes (tasks) along with the security provided by the sites are taken into account. A task model that provides both aspects is used. A discrete QoS levels as a multiple

optional parts of the task are considered, making use of imprecise computation idea. In addition to the QoS and security requirements of the tasks, the communication costs between them are considered.

Since allocation of DAG workload on multiple processors is an NP-hard problem [64], in its basic form, an off-line heuristic allocation algorithm of the workload to produce a suboptimal performance (QoS) of such system is proposed.

To the best of our knowledge this is the first research that considers allocation of dependent real-time tasks with QoS and security requirements on heterogeneous processors to maximize the system QoS while meeting the security constraint of the system and timing constraints of the applications.

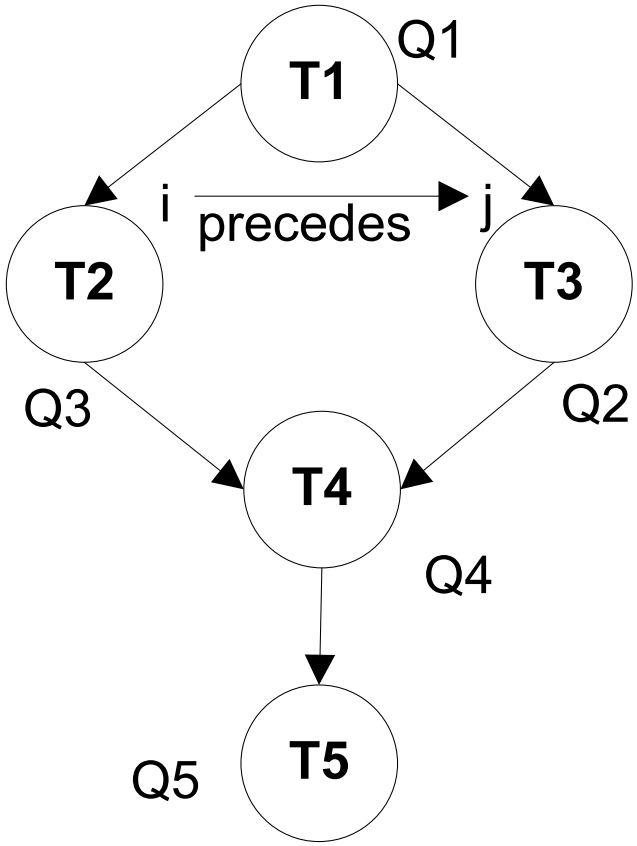
4.4 System Model and Problem Statement

In this section, we give the system and task models; and state the problem along with the performance measures.

4.4.1 System Model

The system model consists of distributed sites and applications modeled as DAG. The site model, see Fig. 2.1, consists of a set, M, of heterogeneous processors and Security Threat Estimator (STE) to monitor the underlying network for any possible security threats. Example of these sites is the UAVs in the previous application scenario. STE sends warning messages to the system in the form of risk level, RL, which is varying over time. RL is the security level the messages should be encoded with to avoid being tampered by a potential eavesdropper in the underlying network.

The application model, Fig. 4.1, is a periodic DAG. Example of an application is the image capturing and subsequent processing tasks on UAV board. The DAG represents one or more combined applications. In a case there is more than one application a comprehensive DAG can be generated. The comprehensive DAG is generated by integration of the application DAGs into one larger DAG by including the instances of the application DAGs up to the Least



(a) Application Model

Figure 4.1 Application Model

Common Multiplier (LCM) of their periods. The method of generating the comprehensive DAG is described in [38] and we do not present its details, due to the space limitations. A comprehensive DAG generation includes the addition of a zero execution time entry and exit nodes if they are not exist and connecting them to existing entry and exit nodes by a zero cost edges. The system is defined as a tuple $G = (V, E, M, T, S, B, D, P)$. The details of the tuple are given below:

- Each vertex $v \in V$ is a task with a set of security levels and a set of QoS levels as will be discussed later.
- $E_{i,j}$ is an edge between vertexes i and j, which represents a precedence relationship, i.e., vertex j cannot start execution before its parents finish their execution. Further, $E_{i,j}$ represents the communication cost per byte of data (time units taken to send a

byte of data), which is dependent on the available communication channel bandwidth. If vertexes i and j are assigned to the same site, the communication cost is assumed to be zero.

- M is the set of heterogeneous processors.
- T is the set of costs $T_{i,l,m}$, that represents the computation times of task $i \in V$ with QoS level l on site m.
- S is the set of costs $S_{k,m}$, which represents the coding/decoding cost of a byte of data in security level k on site m. The level of security should match the risk level (RL) of the underlying network at the time t of message transmission (which is assumed to be the time of message encoding).
- B is the set of data size $B_{i,j}$, which represents the data size that needs to be sent from task i to task j.
- D is the deadline of the DAG which is assumed to be the period.
- P is the period of the DAG.

The DAG contains one entry and one exit nodes. Exit node collects the final processed data from previous nodes for further processing, storing on local storage device or sending to a central server.

4.4.2 Performance Metrics and Measures

The problem can be formally stated as:

"Given a periodic DAG of period P and a set of sites M that are heterogeneous in terms of processing capabilities. Assign tasks to sites and find a feasible schedule, if any, that maximizes the Total QoS Value (TQV) of the system subject to DAG's deadline, communication channel capacity and security risk of using the channel (RL)".

This problem is NP-hard, since the basic allocation of DAG workload on multiple processors is NP-hard [64] (due to the space limitation, we omit the proof). We propose a heuristic algorithm (details are discussed in the next section) to solve this problem. We start by defining some basic parameters that are used in this chapter.

- **Normalized QoS.** The amount of time available for a task i to run can be expressed as a range of QoS levels [52] in $[1, 2, ..., L_i]$, where 1 and L_i are the lowest and highest levels respectively. Each QoS level, l, is given a weight, W_l^q; $0 \leq W_l^q \leq 1$, that reflects its relative importance among other levels. The intuition behind this is that the value returned from using a QoS level will not always increase linearly with the QoS level.

  To capture the importance of any QoS level l for any task i in comparison with other tasks in the system, a normalized level of the QoS is used ($Q_i^l = W_l^q * l/L_i$).

- **Security.** Security, in this research, reflects the length of the encryption strength that is used for securing the data. Other security dimensions like confidentiality, integrity and authentication, which can be combined in a security level, is beyond the scope of this dissertation. Assuming a block encryption algorithm is used in the system, each security level means different encryption key length. For example security level one is mapped to encryption key of 32 bit long and security level two to 64 bit key. The execution time of the security level increases as the level increases. Security levels of the task i are in the range $[1, 2, \ldots, K_i]$, where 1 and K_i are the lowest and highest levels for task i. Security level used to encode any outgoing message on the communication channel should be equal to RL.

- **Earliest Finish Time, EFT_i.** A task, i, in the DAG can only start execution after it has received all messages from its immediate predecessors. Since the sites are heterogeneous, the finish times of the task on different sites are different. Let $EST_{i,m}$ denotes task i earliest start time on site m, $AS_m^{(x)}$ denotes site m available time to execute a new task after finishing task x and $AS_m^{(-x)}$ denotes site m available time before arrival of task x. For a shared channel the channel available time should be taken into consideration.

Let $ACH_i^{(x)}$ denotes the channel available time after the message from task x has been received by task i and $ACH_i^{(-x)}$ denotes the channel available time before receiving the message from task x. For fully connected sites with enough number of channels to avoid contention, the channel available time is not considered and can be omitted in the following equations. Let $Comp_{i,m}$ denotes task i computation time on site m, $predof_i$ denotes the parents set of task i that are not assigned to the same site of i and suc_i denotes the assigned set of task i children, then:

$$EFT_i = \min_{\forall m \in M} \{EST_{i,m} + Comp_{i,m}\} \tag{4.1}$$

$$EST_{i,m} = \max\{\max_{\forall k \in predof_i} \{\max\{EFT_k, ACH_i^{(-k)}\} + E_{k,i}B_{k,i}\}, AS_m^{(-i)}\} \tag{4.2}$$

$$AS_m^{(i)} = \max\{EST_{i,m} + Comp_{i,m}, AS_m^{(-i)}\} \tag{4.3}$$

$$ACH_i^{(x)} = \max\{ACH_i^{(-x)}, EFT_x\} + E_{x,i}B_{x,i} \tag{4.4}$$

$$Comp_{i,m} = T_{i,l,m} + S_{k,m}[\sum_{o \in predof_i} B_{o,i} + \sum_{w \in suc_i} B_{i,w}] \tag{4.5}$$

- **Total QoS Value (TQV).** TQV is our objective, see Eq. 4.7, which is the total QoS resulted from assigning and scheduling the given DAGs on the given sites. The influence of the QoS level of a task output on its successor task's output tends to be the product of QoS levels of both tasks. Intuitively, the task cannot produce a high QoS from a low QoS input. The most a task can do is to improve its input worst-QoS-level where the improvement cannot be more than the input. The interaction between tasks' different QoS levels can be accommodated by assigning weights to each task's output that has a precedence relationship with any other task. Finding those weights is beyond the scope of this dissertation.

  Let $pred_i$ denotes the set of immediate predecessors of task i. Let $I_g = \{I_g^1, I_g^2, ..., I_g^d\}$ denotes the set of d instances for DAG g. Let $Qres_i^d$ denotes the total quality achieved after executing task i of DAG instance d and all of its preceding tasks. Then all total qualities of the tasks are calculated starting from the entry task as given in Eq. 4.6. For any instance d of any DAG g the $Qres_e^d$ at the exit node e gives the QoS value returned

by this particular instance.

$$Qres_i^d = Q_i^c \min_{j \in pred_i} Qres_j^d; \quad \forall i \in I_g^d \tag{4.6}$$

where Q_i^c is the current normalized QoS level of task i and $Qres_1^d$ (for entry node) is Q_1^c.

Let *App* denotes the set of all DAGs in the system. Then the system TQV is the sum of instance's QoS values in the comprehensive DAG.

$$TQV = \sum_{d \in I_g} Qres_e^d; \quad \forall g \in App \tag{4.7}$$

For the DAG in Fig. 4.1, $TQV = \min\{Q1 * Q2, Q1 * Q3\} * Q4 * Q5$.

- **QoS-Degree per Computation time,** $QTCd$**.** It is the ratio of the lowest normalized QoS level to the average computation times of task i QoS levels multiplied by the number of successors. A task with a higher $QTCd$ will, most likely, provide the system with more QoS, because it affects more dependent tasks or/and has more room for improving its QoS.

$$QTCd_i = \frac{Q_i^1}{AVG(E\_Q_i^l)} d_i; \quad 1 \leq l \leq L_i \tag{4.8}$$

where Q_i^1 is the first normalized QoS level, d_i is node's i number of outgoing edges and $E\_Q_i^l$ is the execution time of QoS level l on fastest site (number 1).

- **Success Ratio (**SR**).** It is important to guarantee as many applications as possible. One of the system performance measures is SR, which is the ratio of the admitted number to the total number of applications.

4.4.3 MINLP Formulation

The problem at hand can be formulated as a Mixed Integer Non-Linear Program (MINLP). For the set of sites M; $E_{i,j}$ is the cost of transfer one unit of data (byte) on the communication channel between tasks i and j, S_k denotes the computation time of security for one unit of data on site k, and f_k denotes the computation speed factor of site k. For the set of tasks τ;

$E\_Q_i^q$ denotes the computation time of QoS level q for task i, $B_{i,j}$ is the amount of data units to be sent between task i and j, P is the period of the comprehensive DAG. QL_i denotes the number of QoS levels for task i. D is the set of edges in the comprehensive DAG. s_{i0} is the start time of application's instance that task i belongs to, while s_i is the start time of task i in the comprehensive DAG. d_i is the deadline of the application's instance that task i belongs to. *App* denotes all applications in the comprehensive DAG, $I_g = \{I_g^1, I_g^2, ..., I_g^d\}$ denotes all instances of the application g and ψ_h denotes all possible paths in the application instance h from entry node to exit node.

$p_{i,j}$ is a binary variable that is one if task i precedes task j on the same site otherwise it is zero. $V_{i,k,j,l}$ is a binary variable that is one if task i allocated to site k and task j to site l, otherwise it is zero. $y_{j,k}$ is a binary variable that is one if task j allocated to site k, otherwise it is zero. x_i^q is a binary variable that is one if QoS level q is chosen.

The MINLP is given in Fig. 4.2 which is based on the ILP in [65]. The objective function maximizes the min of the quality values (product of normalized QoS levels of all tasks along a particular path) at the exit node, follows from Eq. 4.7. Constraints (1) and (2) are for the starting time of the tasks. Constraint (3) is for the deadline of the tasks in a DAG. Constraints (4) and (5) are satisfied when either task i preceded task j on the same site or vice versa to ensure that there is no overlapping between tasks on the same site. Constraints (6) and (7) satisfied if and only if one QoS level is chosen and task i is allocated to only one site. Constraints (8), (9), (10) and (11) are satisfied when the tasks are allocated correctly.

4.5 QoS-aware Allocation Heuristic

4.5.1 Application Selection

Originally all tasks are in their lowest QoS level, and all computation times are on the fastest site. Let $Tasks_g$ denotes the set of tasks in the application (DAG) g and P_g denotes the period of DAG g. Then $U_g = (\sum_{j \in Tasks_g} T_{j,1,1})/P_g$ is the utilization of the DAG g considering the tasks in their lowest QoS levels and computation times are on the fastest site (number 1). The applications are sorted in increasing order of the utilizations (U_g). Then

Maximize $TQV = \sum_{g \in App} \sum_{h \in I_g} \min_{a \in \psi_h} \{\prod_{i \in a} \sum_{b \in QL_i} Q_i^b x_i^b\}$
subject to:

Ready time constraints:

$s_i \geq s_{i0}$ $\quad \forall i \in \tau$ 1

$s_j \geq s_i + \sum_{k \in M}[(c_i + \sum_{q \in QL_i} E\_Q_i^q x_i^q) f_k y_{i,k} + \sum_{l \in M}(B_{i,j}(E_{i,j} + S_k)V_{i,k,j,l}) + \sum_{m \in M} B_{w,i} S_k V_{i,k,w,m}]$ $\quad \forall i,j,w \in \tau, ij \in D$ 2

DAG's deadline constraints:

$s_i + \sum_{k \in M}[(c_i + \sum_{q \in QL_i} E\_Q_i^q x_i^q) f_k y_{j,k} + \sum_{l \in M} B_{i,j} S_k V_{i,k,j,l} + \sum_{m \in M} B_{w,i} S_k V_{i,k,w,m}] \leq d_i$ $\quad \forall i,j,w \in \tau, w \neq j$ 3

Tasks' precedence constraints:

$s_j \geq s_i + \sum_{k \in M}[(c_i + \sum_{q \in QL_i} E\_Q_i^q x_i^q) f_k V_{i,k,j,k} + \sum_{l \in M} B_{w,i} S_k V_{i,k,w,l}] - P(1 - p_{i,j})$ $\quad \forall i,j,w \in \tau, i \neq j \ ij, ji \notin D$ 4

$s_i \geq s_j + \sum_{k \in M}[(c_j + \sum_{q \in QL_j} E\_Q_j^q x_j^q) f_k V_{i,k,j,k} + \sum_{l \in M} B_{w,j} S_k V_{j,k,w,l}] - P(p_{i,j})$ $\quad \forall i,j,w \in \tau, i \neq j \ ij, ji \notin D$ 5

Decision variables:

$\sum_{q \in QL_i} x_i^q = 1$ $\quad \forall i \in \tau$ 6

$\sum_{k \in M} y_{i,k} = 1$ $\quad \forall i \in \tau$ 7

$2V_{i,k,j,l} \leq y_{i,k} + y_{j,l}$ $\quad \forall i,j \in \tau, \forall k,l \in M$ 8

$V_{i,k,j,l} - y_{i,k} - y_{j,l} \geq -1$ $\quad \forall i,j \in \tau, \forall k,l \in M$ 9

$2V_{i,k,j,k} \leq y_{i,k} + y_{j,k}$ $\quad \forall i,j \in \tau, \forall k \in M$ 10

$V_{i,k,j,k} - y_{i,k} - y_{j,k} \geq -1$ $\quad \forall i,j \in \tau, \forall k \in M$ 11

$x_i^q, V_{i,k,j,l}, y_{i,k}, p_{i,j} \in \{0,1\}$ $\quad \forall i,j \in \tau, \forall k,l \in M, \forall q \in QL_i$ 12

Figure 4.2 MINLP Formulation of Static DAG Allocation in Distributed Real-time System

Input: Application's tasks in reverse topological order.
Output: Application's tasks with computed LFT.

1: **for** each task i **do**
2: $min \Leftarrow P$
3: **for** each successor s of i **do**
4: **if** $LFT_s - T_{s,1,1} < min$ **then**
5: $min = LFT_s - T_{s,1,1}$
6: **end if**
7: **end for**
8: $LFT_i = min$
9: **end for**

Figure 4.3 Latest Finish Time (LFT) calculation assuming tasks are assigned on fastest site

the QoS-aware allocation algorithm tries to allocate (discussed in the next subsection) all the set of applications. If the allocation is not feasible the allocation is tried after removing the application of greatest utilization and repeating the allocation process. This is repeated till the allocation is feasible or the set of applications becomes empty. At this point the algorithm stops and the resulted allocation, if any, along with TQV is returned. It is worth to mention that for any set of applications, the comprehensive DAG should be built first.

4.5.2 Tasks Allocation

QoS-aware allocation of the tasks includes three main phases: task selection, site selection and selection of the appropriate QoS level of each task to maximize TQV.

4.5.2.1 Task Selection

Task is ready for assignment when all of its predecessors are assigned. Then, priorities are assigned to the ready tasks. The task priority is higher if its Latest Finish Time (LFT) is lower. LFT for each task is calculated upon arrival of its application using the algorithm in Fig. 4.3. The highest priority task is assigned first (ties are randomly broken).

4.5.2.2 Site Selection

The EFT_i (Eq. 4.1) of the selected task i is calculated, and then i is assigned to the site that makes it finishes earlier. If the underlying communication network is contention free then the channel available-time (ACH) is omitted.

4.5.2.3 Maximizing TQV

To maximize the TQV, tasks are assigned to the sites considering lowest QoS levels. Then the TQV maximization scheduling starts.

Therefore we propose EDD_LQL algorithm. LQL means Lowest QoS Levels are used during the assignment phase. In EDD_LQL all the tasks QoS are set to the lowest levels.

After the assignment phase, the actual computation times of the tasks are calculated as in Eq. 4.5, where encoding/decoding of messages are taken care of. The assignment is feasible if each task's EFT is less than its DAG's instance deadline. Note that we take care of the starting time (Eq. 4.2) of the task in the site selection step, otherwise it should be checked and assured to be greater than its DAG's instance start time. Upon feasibility of the allocation, the TQV maximizing process is started as follows:

Let $QTCd\_sorted$ denotes the sorted list of tasks in decreasing order of $QTCd$, see Eq. 4.8. The QoS adaptation process has two steps:

(i) Pick the first task in $QTCd\_sorted$ list,

(ii) For the selected task choose the highest QoS level that does not lead to any deadline miss. The previous steps are repeated for all the tasks. Then TQV is calculated as in Eq. 4.7 ($computeTQV()$ in Fig. 4.4 denotes the TQV calculation process).

The allocation algorithms pseudocode is given in Fig. 4.4.

4.6 Simulation Studies

To evaluate our algorithm, we create a set of random DAGs that is an input to the algorithm. The same set of DAGs is also an input to two baseline algorithms we used for performance comparison with our algorithm. The baseline algorithms are also based on EDDF

Input: All applications (DAGs).
Output: Allocated applications.

```
1: App[ ] = {all applications}
2: while TRUE do
3:    Construct the comprehensive DAG
4:    max = 0
5:    Assign the tasks to sites.
6:    Check feasibility and maximize TQV
7:    if feasible then
8:       max= computeTQV()
9:    end if
10:   if max=0 AND App!=Φ then
11:      App={App}-highest utilization application
12:   else
13:      return
14:   end if
15: end while
```

Figure 4.4 Static DAG Allocation's Algorithm

policy; (i) EDD_MIN uses LQL assignment method and (ii) EDD_MAX uses HQL assignment method. No TQV maximization is applied in EDD_MIN and EDD_MAX.

The random generation of the DAGs is similar to that in [66]. The following parameters are used to generate each DAG used in our study.

- v: *Number of tasks per application.* The number of tasks per application is generated randomly from a uniform distribution with a mean equals to v.

- α: *The shape parameter of the DAG.* The number of levels in the DAG is randomly generated from a uniform distribution with mean value equals to $\alpha\sqrt{v}$. The number of tasks per level is randomly generated from a uniform distribution with mean value equals to $\sqrt{v}/\alpha$. The DAG will show more parallelism if $\alpha \ll 1.0$ and less parallelism if $\alpha \gg 1.0$. Therefore the DAG is balanced if $\alpha = 1.0$.

- β: *Sites heterogeneity factor.* Each site has a computation factor that is generated randomly from a uniform distribution with mean value equal to $\beta/2$. The computation

factor of a site is the ratio of its computation capability to the highest computation capability among sites. As β increases the range of computation factor variation increases and vice versa.

- *CCR: Communication to computation ratio.* The application (DAG) is considered communication extensive if it has a high CCR and vice versa.
- Number of applications is randomly selected from the set { 2,4,6,8,10 }.
- *STA: Number of sites to applications ratio.* As STA increases the number of sites increases. If $STA \gg 1.0$ then the number of sites considered as infinite.
- *numQoS: Number of QoS levels.* Number of QoS levels is generated for each task from a uniform distribution with an average equals to $numQoS$.
- The cost of byte processing by local and remote communication subsystem, and transfer over the network, TB.
- *avgcomp: The average computation time of the application.* The tasks computation times are generated from a uniform distribution with a mean equals to $avgcomp$. The average communication cost is calculated as $CCR * avgcomp$.
- *$succNum$: The number of successors.* Each task in the DAG has a number of successors that is generated from a uniform distribution with an average equals to $succNum$.
- *SF: Computation factor of the security per byte of the message.* SF is generated for each site from a uniform distribution with an average equals to $\beta * 0.01/2$.
- *P: Period of the application.* Each generated DAG has an end-to-end deadline that is the same as its period. P is randomly selected from the set {66.67, 50, 40, 33.33}, to simplify calculations of the LCM (Least Common Multiplier), which is equals to 200 in this case.

The above input parameters were varied over the following:

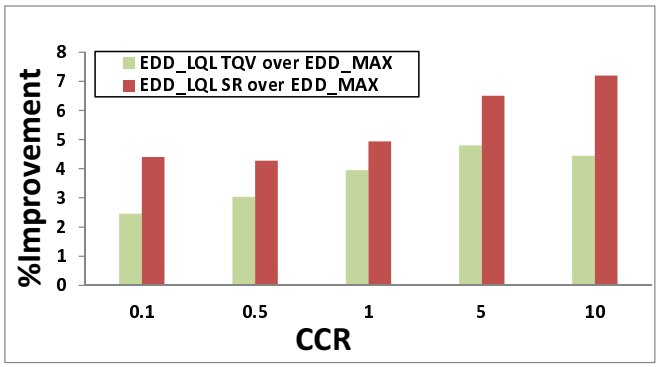
- $v = \{20, 40, 60, 80, 100\}$
- $\alpha = \{0.5, 1.0, 2.0\}$
- $\beta = \{0.1, 0.25, 0.5, 1.0\}$
- $CCR = \{0.1, 0.5, 1.0, 5.0, 10\}$
- $STA = \{0.25, 0.5, 1.0, 2.0, 4.0, 10.0\}$
- $numQoS = \{5\}$
- $TB = \{0.01\}$
- $avgcomp = \{5\}$
- $succNum = \{1, 4, 5, 10, 50, 100\}$

Based on the above values, a (10,800) different DAGs are generated. For each combination of the above parameters a DAG with the same average parameters and a random period is generated 10 times. Therefore we have a total of 108,000 DAGs. We assumed that the security levels of all tasks are constant and equal to RL. Performance investigation under dynamic variation in RL is reserved as a future work.

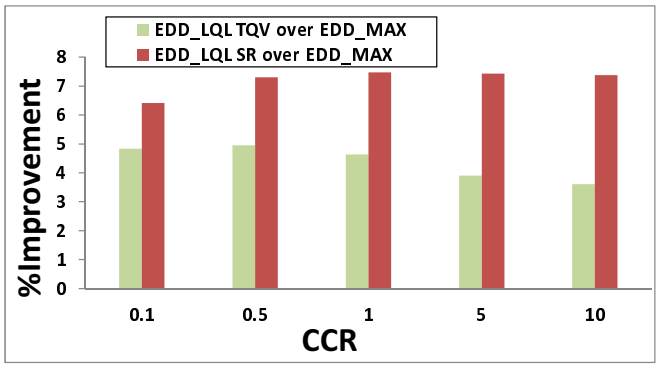
We studied the impacts of several parameters on the performance of the EDD_LQL algorithm for contention free (fully connected) and shared channels. Two important parameters impacts are shown in the following. Further, we studied the impact of the applications rejection criteria and the system usage by the proposed algorithms.

4.6.1 Impact of CCR

For contention free channels, the performance decreases as CCR increases. Because communication costs use up most of the time available for admitting the applications, communication cost is considered as the bottleneck in this case. As the applications becomes communication extensive ($CCR \geq 5$), the performance tends to be saturated. But the improvements of EDD_LQL over EDD_MAX, Fig. 4.5(a), in TQV and SR tend to increase as CCR increases.



(a) Improvements for contention free channels

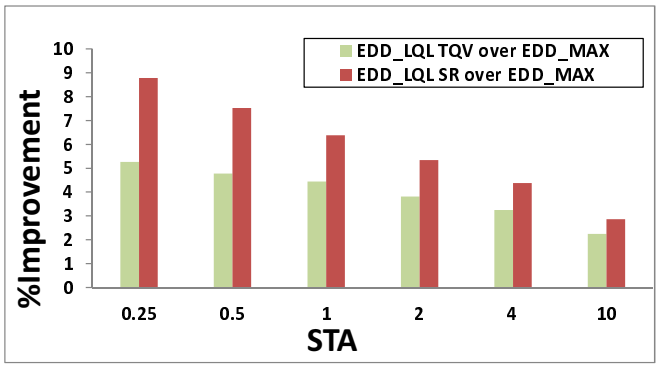


(b) Improvements for shared channel

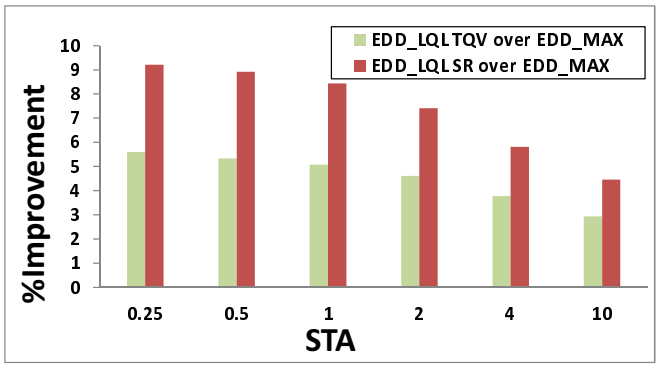
Figure 4.5 Improvements in TQV and SR vs. CCR for shared and contention free channels

That is connected to a larger number of accepted applications by EDD_LQL as compared with EDD_MAX and the ability of EDD_LQL to utilize the available room on sites to increase DAG's QoS levels.

For shared channel, the performance tends to take a convex shape where the best performance is in the interval [0.5, 1]. When CCR is low the algorithms distribute tasks on more sites, which makes the precedence constraints more effective in reducing the number of admitted applications and hence low TQV. As CCR increases the effect of precedence constraint gets lower, because communicating tasks tend to be assigned to same sites. This makes the algorithms more effective in increasing the TQV and SR. However more increase in CCR limits any use of parallelism in the applications which again limits the number of admitted applications and hence a drop in performance. The improvements of EDD_LQL over EDD_MAX,



(a) Improvements for contention free channels



(b) Improvements for shared channel

Figure 4.6 Improvements in TQV and SR vs. STA for shared and contention free channels

Fig. 4.5(b), in SR tends to increase as CCR increases up to one. Then the improvement is saturated due to the fixed number of admitted applications. The improvement in TQV takes a concave shape that declines as CCR increases because most of the time slack in sites is consumed for communication purposes.

4.6.2 Impact of STA

For contention free channel case, as the number of available sites decreases the improvement in terms of TQV and SR increases which emphasis the power of our adaptive algorithm when the resources are scarce. Where as in shared channel case the improvement follows the same trend except when STA is 0.25. When STA is 0.25 the number of sites is very limited and not more than two in most of the cases. Hence the room for improvement is not enough to keep

the same trend.

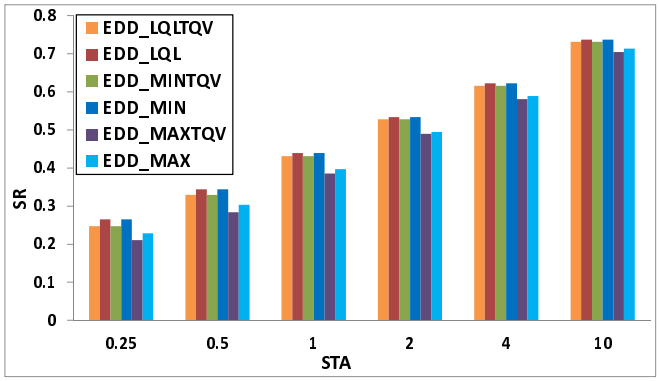
4.6.3 Impact of the Applications Rejection Criteria

The application with the greatest utilization among the arriving applications is rejected when the algorithm is not able to admit all of them. This rejection criterion, which is used in the proposed and baseline algorithms, is considered because the application with greatest utilization will most likely leave more room on the sites for other applications to be admitted. Other criteria for applications' rejection could be considered. An application with the lowest priority is considered for rejection before other higher priority applications. To evaluate the effectiveness of utilization rejection criterion, a rejection criterion that is based on the quality per number of application instances is used. An application with the lowest quality-per-instances value is rejected first. The same rejection criterion is used for all algorithms where the applications are assumed to have the same priorities.

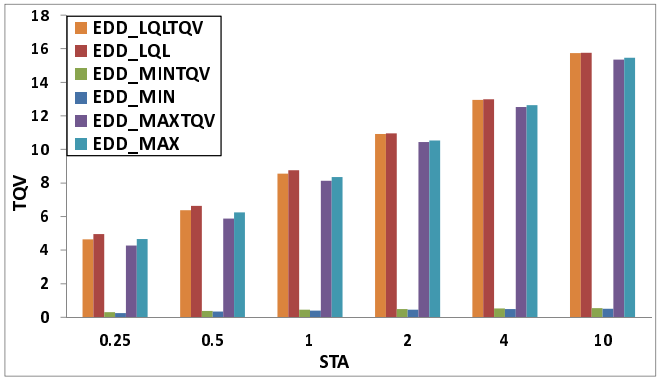
Fig. 4.7 shows the performance of the algorithms using utilization rejection criterion and quality rejection criterion. For the version of the algorithm that uses quality as the rejection criterion, TQV is appended to the name of the algorithm that uses utilization criterion for rejection. For SR and TQV the algorithms that are using utilization as a rejection criterion perform better than when the rejection criterion is based on quality per number of instances of the application. This performance is expected because the application with the lowest quality-by-number-of-instances value is not guaranteed to leave enough room for other applications to be accepted. Quality-by-number-of-instances rejection criterion favors quality over admittance of more applications to the system which is more important in the admittance phase of applications.

4.6.4 System Usage Study

To study the systems' resources usage by allocated tasks as the average load per site, the sites usage was logged during the simulation experiment in the previous sub-section. Since the sites are heterogeneous in computational capabilities, the sites' capabilities needs firstly to



(a) SR vs. STA



(b) TQV vs. STA

Figure 4.7 SR and TQV vs. STA for contention free channels

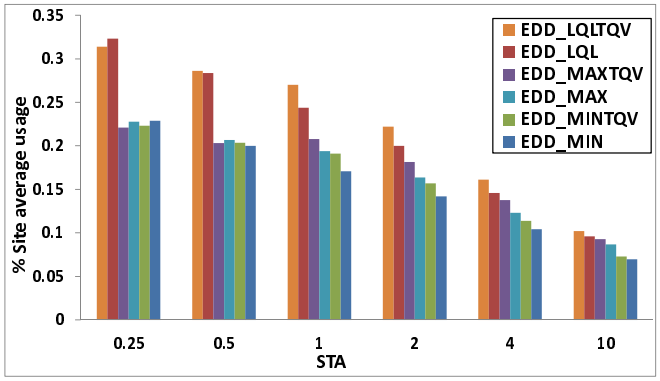


Figure 4.8 Percentage of site usage in the system vs. STA for contention free channels

be normalized to the lowest computational capability site. The lowest site in computational capabilities is assumed to be the site with computation factor of one. Therefore, each site is multiplied by a weight that equals the reciprocal of its computation factor to get its normalized computation capability. The number of sites in the system are considered equals to the number of lowest-computational-capability sites that have the same computational power (capability) of all other sites. In other words, each site equals reciprocal-of-its-computation-factor lowest computation factors sites. Then, the sum of normalized system usage is divided by the number of the sites in the system to get the average system usage per site.

Fig. 4.8 shows the percentage of system usage per site vs. the STA. As the STA increases the system usage decreases due to availability of resources. The proposed algorithms (EDD_LQL and EDD_LTQV) have the highest system usage per site, because these algorithms modify the tasks computation cost to maximize the TQV of the system. Hence, the ability to utilize more resources is better than other algorithms.

Although the objective of our work is not to have a fair usage between sites, the system usage result, under the simulated conditions, shows that there is a merit for improvements. For other simulations conditions the results may be different. Improvements can be in developing algorithms that provide more fairness usage of the sites and in developing more efficient algorithms that utilize the unused holes in the tasks schedule in the system.

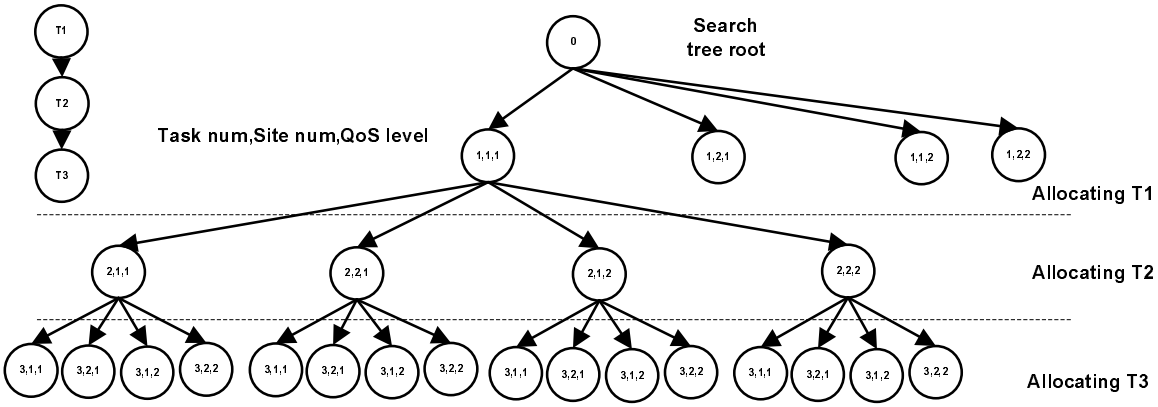


Figure 4.9 Search tree of assigning three tasks (each with two QoS levels) to two sites

4.7 Branch and Bound Heuristic Algorithm

The B&B algorithm can be used to find a solution to the problem at hand by exploring as small search space as possible. To optimally allocate tasks of the DAG to the available sites, all the possibilities should be considered. Therefore a search tree should be used to enumerate all the solutions and take the optimal (best solution). Fig. 4.9 shows the search tree for assigning three tasks to two sites where each of the tasks has two QoS levels. An expansion of the node (Task 1,Site 1,QoS level 1) is shown in the figure. Other nodes have the same expanded set of nodes.

A B&B algorithm is used to allocate a workload modeled as DAG on a given heterogeneous sites. The algorithm proceeds in iterations. In each iteration the algorithm processes only one node in the search tree which initially only contains the root. The iteration has three basic steps; node selection, branching and bound calculation.

4.7.1 Node Selection and Branching

The algorithm starts on the root node which has a higher bound of TQV that is zero, and creates its children, which are all possible assignments of the first level tasks to sites. For n tasks in a level k to be assigned to m sites, where each task has q QoS levels, a total of $n.m.q$ possibilities are needed (see Fig. 4.9). Then the application performance, i.e., the TQV, is calculated under the allocation branch descending from each child. The TQV calculation

includes the calculation of TQV for the assigned tasks that are already allocated and the tasks to be allocated, which implies an estimation of the non-assigned-tasks quality improvement. The creation of children proceeds for the child with highest TQV, because this is the higher bound of TQV for the allocations descending from this branch. The children that lead to a non-feasible allocation will be pruned, while newly created children with TQVs less than highest TQV will be saved in a set of active nodes to be explored if the current branch did not provide a feasible allocation. This procedure will give the first solution if one exist, because it uses exhaustive search to find it, but guided by the cost calculation algorithm that runs in polynomial time which will restrict the search space. This is the famous Branch and Bound (B&B) algorithm. The version that we use in our allocation algorithm is given in Fig. 4.10.

4.7.2 TQV Upper Bound Calculation

The key step in the algorithm is to calculate the upper bound for the allocations descending from the current vertex, and then to branch from the vertex with highest estimated TQV. The Critical Path (CP) in a DAG is the set of tasks on a path from entry to exit nodes that incurs the greatest sum of computations' cost. The number of sites are assumed to be unlimited for the sake of the non-assigned tasks' allocation. Finish time of the last task on the CP gives the minimum finish time for the DAG, considering tasks in their highest QoS levels. The TQV is calculated and considered the upper bound of TQV for a feasible allocations descending from this vertex. If a leaf node is reached, the calculated TQV is exact. At this point all active nodes with TQV less than the exact will be pruned. let $Comp_i$ denotes the computation cost of task i on its site, see Eq. 4.5. Let r_{i0} denotes the start time of application's instance that task i belongs to. The TQV calculation from vertex v (denoted by $getTQV(v)$ in the algorithm in Fig. 4.10) proceeds in the following steps:

S1. For all tasks in the DAG compute ready times as follows:

- If task i is assigned, then communication cost from preceding tasks $j \in predof_i$, is considered, where the cost is zero if both tasks are assigned to the same site. The

```
1: Active set A={root}
2: maxTQV ⇐ getTQV(root)
3: while TRUE do
4:    Let V is the vertex of maximum TQV
5:    if V is a leaf vertex then
6:       Prune all v ≠ V ∈ A
7:       Return V //solution
8:    else
9:       Generate children of V
10:      A = A − {V}
11:      for all Children c of V do
12:         if getTQV(c) > 0 /*feasible*/ then
13:            A = A ∩ c
14:            if  c is a leaf vertex then
15:               if getTQV(c) > maxTQV then
16:                  maxTQV = getTQV(c)
17:                  Prune all v ≠ c ∈ A
18:                  Return c //solution
19:               else
20:                  Prune all v ≠ c ∈ A with TQV_v < maxTQV
21:               end if
22:            end if
23:         end if
24:      end for
25:   end if
26:   if A is empty then
27:      return 0// no solution
28:   end if
29: end while
```

Figure 4.10 Static assignment and scheduling of a DAG in distributed system using B&B algorithm

task ready time is:

$$r_i = max\{\max_{j \in predof_i}[r_j + Comp_j + E_{j,i}B_{j,i}], r_{i0}\} \tag{4.9}$$

- If task i is not assigned, then the communication cost is not considered. Then task i ready time is:

$$r_i = max \max_{j \in predof_i}[r_j + Comp_j, r_{i0}] \tag{4.10}$$

S2. If finish time of the last task on the CP of the DAG or the CP of any applications' instance is larger than the DAG's deadline or the instance's deadline then return zero (not feasible). Otherwise increase QoS of the non-assigned tasks sorted in decreasing order of QTCd value to the possible maximum level.

S3. Calculate TQV for the DAG (comprehensive DAG) as given in Eq.4.7 and call it TQV_v.

When exploring a vertex, the starting time and computation cost, for each task i on site j are logged. The allocation and scheduling of tasks can be obtained by backtracking the allocations' time slots from the leaf node where actual assignment is done.

The nature of the problem at hand prevents the B&B algorithm to produce the optimal solution. The problem at hand is non-linear which result in skipping some nodes that may lead to an optimal solution.

The B&B algorithm simulation results are not presented in this dissertation.

4.8 Conclusions

In this chapter, the problem of security and QoS-aware application assignment and scheduling on heterogeneous distributed real-time systems was addressed. A heuristic algorithm to solve the problem for shared and contention free communication channels was proposed. The proposed algorithm was evaluated by extensive simulation experiments using wide range of randomly generated workload. The evaluation showed that the proposed algorithm outperformed the baseline algorithms in all cases. For contention free channels the performance magnitude is higher than that for shared channel case. However the difference in magnitude is

not very significant. A Branch and Bound (B&B) heuristic procedure was described to solve the problem by exploring only a subset of the solutions space.

CHAPTER 5 Dynamic DAG Allocation in Distributed Real-Time Systems

5.1 Summary

We consider, in this chapter, a set of dependent real-time tasks, modeled as Directed Acyclic Graph (DAG), with security and QoS requirements for a dynamic assignment and scheduling on a set of heterogeneous sites with the objective of maximizing Success Ratio (SR) and Total Quality Value (TQV) of the system.

Distributed real-time systems with heterogeneous nodes in terms of processing capabilities are continuously evolving to realize many emerging mission critical applications, e.g., battle field vision systems. In such systems, a tradeoff exists between quality of results and security of task execution. In this chapter, the work in Chapter 4 is extended to address the dynamic assignment and scheduling of dependent tasks with QoS and security requirements on heterogeneous system. In particular we make the following contributions; (i) propose two polynomial time heuristic algorithms to maximize SR and then TQV, (ii) a proof of concept, using simple experiment on InfoSphere platform, is provided, and (iii) we evaluate the algorithms through simulation studies by comparing it to baseline algorithms for variations of synthetic workloads. The proposed algorithms outperform the baseline algorithms in all the simulated conditions for fully-connected and shared bus network topologies.

5.2 Background

In Chapter 4, the static allocation of dependent tasks with QoS and security requirements on heterogeneous sites was studied. However the workload in such systems is dynamic in na-

ture, meaning tasks arrive when there is a service required to be executed and then leave when that service is done. In this chapter, we extend this work to address the allocation problem of dynamically arrived precedence constrained-tasks to heterogeneous sites in a distributed security-sensitive system. In particular; the problem to address the dynamics of tasks is formulated, and then new algorithms that take into account fast allocation of tasks on sites are developed and the proposed algorithms are evaluated using extensive simulation experiments.

The goal is to find a feasible allocation of tasks to sites that maximize the guarantee ratio (number of admitted applications to the total number of arrived applications), and maximizes the objective value (i.e., TQV, to be described later). By feasible allocation, we mean to find a feasible schedule under a given assignment that meets the constraints of timing, precedence and security.

The rest of this chapter is organized as follows. the related literature is provided in Section 5.3. Problem statement and review of the system model, in [67], are given in Section 5.4. In Section 5.5 the QoS-aware allocation algorithm is discussed. A proof of concept, using simple experiment on InfoSphere platform, is provided in Section 5.6. Simulation studies are shown and discussed in Section 5.7. Finally, Section 5.8 concludes the chapter.

5.3 Related Work

DAG's allocation on a set of distributed processors has been extensively studied by the research community. There are two general methods to assign DAGs to distributed processors; clustering based and list based. In the clustering based assignment, tasks are clustered according to the ratio of the communication cost to the computation cost of the communicating pairs of tasks [38]. In the list based assignment, tasks are assigned according to their priorities [39]. The priority of a task reflects its important. In both types of assignments the scheduling stage is taken place after the assignment. The survey [41] gives a detailed discussion of DAG's assignment and scheduling on distributed systems.

In [39] the authors used a list based assignment policy. They addressed the effect of error in an input to a task in the DAG after partially completed preceded tasks. They proposed a

modified versions of well known algorithms to dynamically allocate tasks on a homogeneous distributed real-time system. In this chapter, EDF (EDDF) list assignment policy is used. Processors (sites) are assumed to be heterogeneous and the security constraint of the underlying network is taken into account.

Further, Security and QoS issues have been addressed by many researchers when studying resource management in time critical systems. Some of the researchers addressed QoS [32, 39, 60, 61]. Others studied security [17, 19, 30, 31]. The researchers who studied both security and QoS, they either restricted their study on single processor system and assuming the tasks are preemptable [33], or assumed the tasks are independent when studied distributed systems [18,63]. Some studied QoS optimization for independent tasks in real-time clusters; to support the fault tolerance of the system [26] or to produce a fair QoS level between the admitted tasks and maximize the schedulability [68].

The problem of on-line allocation of a set of periodic Directed Acyclic Graphs (DAGs) on a set of heterogeneous sites (processors) is addressed. In this chapter, the QoS and security requirements of the nodes (tasks) along with the security provided by the sites are considered. A task model that provides both aspects is used. A discrete QoS levels as a multiple optional parts of the task or multiple versions of the task are used, making use of imprecise computation idea. In addition to the QoS and security requirements of the tasks, the communication costs between them are considered.

Since allocation of DAG workload on multiple processors is an NP-hard problem [64], in its basic form, we propose an on-line heuristic allocation algorithm of the workload to produce a suboptimal performance (QoS) of such system.

To the best of our knowledge this is the first research that considers on-line allocation of dependent real-time tasks with QoS and security requirements on heterogeneous processors to maximize the system QoS while meeting the security constraint of the system and timing constraints of the applications.

5.4 System Model and Problem Statement

In this section, the system and task models; and the performance measures are reviewed as given in [67]. Then the on-line allocation problem of a set of DAGs on heterogeneous sites is stated.

5.4.1 System Model

The system model consists of distributed sites and applications modeled as DAG. The site model, see Fig. 2.1, consists of a set, M, of heterogeneous processors and Security Threat Estimator (STE) to monitor the underlying network for any possible security threats. STE sends warning messages to the system in the form of risk level, RL, when it finds any security breaches in the underlying network. RL is the security level the messages should be encoded with to avoid being tampered by a potential eavesdropper in the underlying network.

The applications model, Fig. 4.1, is a periodic DAG. Example of an application is the image capturing and subsequent processing tasks that can take place on sites (e.g., UAV board). One possible mission of a UAV team is to recognize/detect targets in the battle field, Fig. 2.4, where each of the images taken is segmented (using one of many available algorithms) to several segments then a classification stages (using one of several available methods) are taken place where results are combined to take a suitable decision which might be slowing or speeding up the image capturing rate. Details of the application are beyond the scope of this dissertation.

The DAG represents one application. In case there is more than one application a comprehensive DAG can be generated. The comprehensive DAG is generated by integration of the application DAGs into one larger DAG by including the instances of the application DAGs up to the double of Least Common Multiplier (LCM) of their periods plus largest period of the DAGs [54]. The method of generating the comprehensive DAG is described in [38] and we do not present its details, due to the space limitations. A comprehensive DAG generation includes the addition of a zero execution time entry and exit nodes if they are not exist and connecting them to existing entry and exit nodes by a zero cost edges. The system is defined

as a tuple $G = (V, E, M, T, S, B, D, P, A)$. The details of the tuple are given below:

- Each vertex $v \in V$ is a task with a set of security levels and a set of QoS levels as will be discussed later.
- $E_{i,j}$ is an edge between vertexes i and j, which represents a precedence relationship, i.e., vertex j cannot start execution before its parents finish their execution. Further, $E_{i,j}$ represents the communication cost per byte of data (time units taken to send a byte of data), which is dependent on the available communication channel bandwidth. If vertexes i and j are assigned to the same site, the communication cost is assumed to be zero.
- M is the set of heterogeneous processors.
- T is the set of costs $T_{i,l,m}$, that represents the computation times of task $i \in V$ with QoS level l on site m.
- S is the set of costs $S_{k,m}$, which represents the coding/decoding cost of a byte of data in security level k on site m. The level of security should match the risk level (RL) of the underlying network at the time of message transmission (which is assumed to be the time of message encoding).
- B is the set of data size $B_{i,j}$, which represents the data size that needs to be sent from task i to task j.
- D is the deadline of the DAG which is assumed to be the period.
- P is the period of the DAG.
- A is the arrival time of the application.

The DAG contains one entry and one exit nodes. Exit node collects the final processed data from previous nodes for further processing, storing on local storage device or sending to a central server.

5.4.2 Performance Metrics and Measures

The problem can be formally stated as:

"Given a periodic DAGs each of period $p \in P$***, arrival time*** $a \in A$ ***and a set of sites*** M ***that are heterogeneous in terms of processing capabilities. Assign as many applications as possible to sites and find a feasible schedule, if any that maximizes the Total QoS Value (TQV) of the system subject to DAG's deadline, communication channel capacity and security risk of using the channel (RL))"***.

This problem is NP-hard, since the basic allocation of DAG workload on multiple processors is NP-hard [64] (due to the space limitation, we omit the proof). We propose a set of heuristic algorithms (details are discussed in the next section) to solve this problem. We provide a definition of basic parameters that are used in this chapter.

- **Normalized QoS.** The amount of time available for a task i to run can be expressed as a range of QoS levels [52] in $[1, 2, ..., L_i]$, where 1 and L_i are the lowest and highest levels respectively. Each QoS level, l, is given a weight, W_l^q; $0 \leq W_l^q \leq 1$, that reflects its relative importance among other levels. The intuition behind this is that the value returned from using a QoS level will not always increase linearly with the QoS level.

  To capture the importance of any QoS level l for any task i in comparison with other tasks in the system, a normalized level of the QoS is used ($Q_i^l = W_l^q * l/L_i$).

- **Security.** Security reflects the length of the encryption strength that is used for securing the data. The execution time of the security level increases as the level increases. Security levels of the task i are in the range $[1, 2, \ldots, K_i]$, where 1 and K_i are the lowest and highest levels for task i. Security level used to encode any outgoing message on the communication channel should be equal to RL.

- **Earliest Finish Time, EFT_i.** A task, i, in the DAG can only start execution after it has received all messages from its immediate predecessors. Since the sites are heterogeneous and the data volumes sent/received by communicating tasks are not the same, the finish times of the task on different sites are different. Let $EST_{i,m}$ denotes task i earliest start

time on site m, $AS_m^{(x)}$ denotes site m available time to execute a new task after finishing task x and $AS_m^{(-x)}$ denotes site m available time before arrival of task x. For a shared channel the channel available time should be taken into consideration. Let $ACH_i^{(x)}$ denotes the channel available time after the message from task x has been received by task i and $ACH_i^{(-x)}$ denotes the channel available time before receiving the message from task x. For fully connected sites with enough number of channels on each link between sites to avoid contention, the channel available time is not considered and can be omitted in the following equations. Let $Comp_{i,m}$ denotes task i computation time on site m, $predof_i$ denotes the parents set of task i that are not assigned to the same site of i and suc_i denotes the set of task i children that are not assigned to the same site of i, then:

$$EFT_i = \min_{\forall m \in M} \{EST_{i,m} + Comp_{i,m}\} \tag{5.1}$$

$$EST_{i,m} = \max\{\max_{\forall k \in predof_i} \{\max\{EFT_k, ACH_i^{(-k)}\} + E_{k,i}B_{k,i}\}, AS_m^{(-i)}\} \tag{5.2}$$

$$AS_m^{(i)} = \max\{EST_{i,m} + Comp_{i,m}, AS_m^{(-i)}\} \tag{5.3}$$

$$ACH_i^{(x)} = \max\{ACH_i^{(-x)}, EFT_x\} + E_{x,i}B_{x,i} \tag{5.4}$$

$$Comp_{i,m} = T_{i,l,m} + S_{k,m}[\sum_{o \in predof_i} B_{o,i} + \sum_{w \in suc_i} B_{i,w}] \tag{5.5}$$

- **Total QoS Value (TQV).** TQV is our objective, see Eq. 5.7, which is the total QoS resulted from assigning and scheduling the given DAGs on the given sites. The influence of the QoS level of a task output on its successor task's output tends to be the product of normalized QoS levels of both tasks. Intuitively, the task cannot produce a higher QoS from a low QoS input. The most a task can do is to improve its input worst-QoS-level where the improvement cannot be more than the input. The interaction between tasks' different QoS levels can be accommodated by assigning weights to each task's output that has a precedence relationship with any other task. Finding the weights is beyond the scope of this research.

  Let $pred_i$ denotes the set of immediate predecessors of task i. Let $I_g = \{I_g^1, I_g^2, ..., I_g^d\}$

denotes the set of d instances for DAG g. Let $Qres_i^d$ denotes the total quality achieved after executing task i of DAG instance d and all of its preceding tasks. Then all total qualities of the tasks are calculated starting from the entry task as given in Eq. 5.6. For any instance d of any DAG g the $Qres_e^d$ at the exit node e gives the QoS value returned by this particular instance.

$$Qres_i^d = Q_i^c \min_{j \in pred_i} Qres_j^d; \quad \forall i \in I_g^d \tag{5.6}$$

where Q_i^c is the current normalized QoS level of task i and $Qres_1^d$ (for entry node) is Q_1^c. Let App denotes the set of all DAGs in the system. Then the system TQV is the sum of instance's QoS values in the comprehensive DAG.

$$TQV = \sum_{d \in I_g} Qres_e^d; \quad \forall g \in App \tag{5.7}$$

For the DAG in Fig. 4.1, $TQV = \min\{Q1 * Q2, Q1 * Q3\} * Q4 * Q5$.

- **QoS-Degree per Computation time, $QTCd$.** It is the ratio of the lowest normalized QoS level to the average computation times of task i QoS levels multiplied by the number of successors and predecessors. A task with a higher $QTCd$ will, most likely, provide the system with higher QoS, because it affects more dependent tasks or/and has more room for improving its QoS.

$$QTCd_i = \frac{Q_i^1}{AVG(E\_Q_i^l)} d_i; \quad 1 \leq l \leq L_i \tag{5.8}$$

where Q_i^1 is the first normalized QoS level, d_i is node's i number of ingoing and outgoing edges; and $E\_Q_i^l$ is the execution time of QoS level l on fastest site (i.e., site 1).

- **Success Ratio (SR).** It is important to guarantee as many applications as possible. One of the system performance measures is SR, which is the ratio of the admitted number to the total number of applications.

5.5 QoS-aware Allocation Heuristics

Based on the dynamics of the workload, we propose two slightly different algorithms to allocate tasks to sites:

- $TQVS\_S$. This algorithm assigns all the application (DAG) instances to the same sites of the first instance. Tasks of already assigned applications will not be moved to another site. This algorithm eliminates task movement cost when already assigned task is moved to another site.

- $TQVS\_D$. All tasks; the assigned and the newly arrived, can be assigned anywhere without any restrictions. This algorithm is useful when movement of tasks between sites has no or low cost.

The allocation algorithm has two integrated phases; the admission phase and the allocation phase.

5.5.1 Application Admission

If more than one application arrived at the same time then the applications are sorted in increasing order of the utilizations (U_g). $U_g = (\sum_{j \in T_g} T_{j,1,1})/P_g$ is the utilization of the DAG g considering the tasks in their lowest QoS levels and computation times are on the fastest site. Where T_g denotes the set of tasks in the application (DAG) g and P_g denotes the period of DAG g. Then the admission control module builds a comprehensive DAG (up to 2LCM + max{periods} [54] or up to the life time of the application which one is the smallest) that includes all applications already admitted to the system and the newly arrived applications. The admission controller then works on the comprehensive DAG and allocates all tasks to sites as if the sites are free of loads except the running tasks.

Then the controller, after allocating arrived application, checks for any deadline miss. If there is any deadline miss then the allocation is not feasible. Therefore the admission module repeats the admission process after removing the application of greatest utilization. This is repeated till the allocation is feasible. Only the applications left in the list are admitted to

the system. The list of arrived applications can be empty and hence no application can be admitted.

5.5.2 Tasks Allocation

QoS-aware allocation of the tasks includes three main phases: task selection, site selection and selection of the appropriate QoS level of each task to maximize TQV.

5.5.2.1 Task Selection

Task selection phase is the same of our previous work [67]. We repeat the procedure here for completeness of the discussion. Task is ready for assignment when all of its predecessors are assigned. Then, priorities are assigned to the ready tasks. The task priority is higher if its Latest Finish Time (LFT) is lower. LFT for each task is calculated upon arrival of its application using the algorithm in Fig. 5.1. The highest priority task is assigned first (ties are randomly broken).

Input: Application's tasks in reverse topological order.
Output: Application's tasks with computed LFT.

1: **for** each task i **do**
2: $min \Leftarrow P$
3: **for** each successor s of i **do**
4: **if** $LFT_s - T_{s,1,1} < min$ **then**
5: $min = LFT_s - T_{s,1,1}$
6: **end if**
7: **end for**
8: $LFT_i = min$
9: **end for**

Figure 5.1 Latest Finish Time (LFT) calculation assuming tasks are assigned on fastest site

5.5.2.2 Site Selection

In this phase, assignment can be one of two types:

- Already assigned tasks cannot be reassigned to different sites and once the first instance is assigned for the arrived application(s) other instances take the same assignment.
- All the tasks of already assigned and of the arrived applications are considered as not assigned and then the assignment is started where each task can be assigned to any site regardless of its application instance.

In both types, exact security cost can be taken into account which implied recalculating and rearranging tasks starting and execution times on sites on each task assigning trial.

The EFT_i calculation as given in Eq. 5.1 can be implemented using dynamic programming. However we implemented it accumulatively in two steps; (i) find computation time of the current task i, (ii) calculate starting time of the current task i and update site's ready time. These two steps are given in Fig. 5.2.

Input: Tasks in the assignment order up to current task i.
Output: Application's tasks with computed timings.

1: **for** each unassigned task i **do**
2: Sum up all data from/to assigned parents/children on other sites.
3: Add Decoding/Encoding cost of data sum to computation cost of i on its site m.
4: **end for**
5: **for** each task i **do**
6: Find EFT_i of i on its site m using Eq. 5.1.
7: Update $AS_m \leftarrow EFT_i$.
8: **end for**

Figure 5.2 Timings calculation of the applications in the system

If the underlying communication network is contention free then the channel available-time (ACH) is omitted.

5.5.2.3 Maximizing TQV

To maximize the TQV, tasks are assigned to the sites considering lowest QoS levels. Then the TQV maximization process starts.

In all of the proposed algorithms, the tasks QoS are set to the lowest levels, and then the assignment of tasks to sites is conducted. The assignment is feasible if each task's EFT is smaller than its DAG's instance deadline. Note that we take care of the starting time (Eq. 5.2) of the task in the site selection step, otherwise it should be checked and assured to be greater than its DAG's instance start time. Upon feasibility of the allocation, the TQV maximizing process is started as follows (the same as in [67]:

Let $QTCd\_sorted$ denotes the sorted list of tasks in decreasing order of $QTCd$, see Eq. 5.8. The QoS adaptation process has two steps:

(i) Pick the first task in $QTCd\_sorted$ list,

(ii) For the selected task choose the highest QoS level that does not lead to any deadline miss. The previous steps are repeated for all the tasks. Then TQV is calculated as in Eq. 5.7 ($computeTQV()$ in Fig. 5.3 denotes the TQV calculation process).

The allocation/admission algorithm pseudocode is given in Fig. 5.3.

5.6 A Proof of Concept on InfoSphere Platform

InfoShpere platform, [2], is a stream processing middleware which is developed and maintained by IBM Corporation. Data streams are continuous flows of data [69]. Examples of data streams include network traffic, sensor data and call center records. Stream processing system is different from one query system. One query system is static, where it deals with the data for one time by sending a query to the system and waits for the response. In streaming system, on the other hand, the query is composed of repeated queries. Fig. 5.4 shows the difference between both concepts.

InfoSphere platform provides tools and built in operators (processing elements; PE) that help developers to design basic functionalities of their applications. For example, Join operator correlates contents of several streams based on user criteria. Custom operators can also be built by user, besides other operators that are available in InfoSphere, e.g., Sort, Filter, Delay, etc.

To run an application on InfoSphere middleware, one needs to have an account in the

Input: All applications; admitted and arrived.
Output: Allocated applications.

1: $App[\,] = \{all\ arrived\ applications\}$
2: **while** TRUE **do**
3: Construct the comprehensive DAG of admitted and arrived applications
4: $max = 0$
5: **for** each task i in the comprehensive DAG **do**
6: Assign i to the suitable site (According to $TQVS\_S$ or $TQVS\_D$).
7: Use algorithm in Fig. 5.2 to find computation and starting times of i; and to update site available time.
8: **end for**
9: Check feasibility.
10: **if** feasible **then**
11: Maximize TQV.
12: max= computeTQV().
13: **end if**
14: **if** max=0 AND App!=Φ **then**
15: App={*App*}-{highest utilization application}
16: **else**
17: return
18: **end if**
19: **end while**

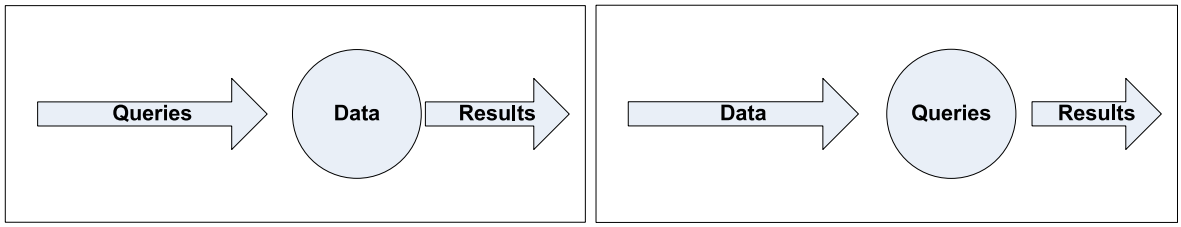
Figure 5.3 Dynamic DAG Allocation's Algorithm

system. After logging in to the system, an instance (autonomous unit that is consists of several nodes) should be created or a previously created instance should be used to submit jobs for running on the platform. More details are available from IBM information center [70].

We build a simple application on InfoSphere platform [71] to show how dynamic-adaptation of QoS helps in improving the schedulability of the system. This application is shown in Fig. 5.5.

The sample application consists of three main tasks: (i) *Image Capture* that takes images using a webcam, (ii) *Edge Detection* has three versions where each one runs an edge detection algorithm on the received frame. The execution time for high quality version is longer than for low quality version, (iii) *Save Image* task saves the received edge-detected frame.

Feedback control is added to the application to control the frame rates by varying the version used in *Edge Detection* task through the feedback message that is piggybacked on the image



(a) Static (b) Streaming system

Figure 5.4 Static Vs. Streaming concept [2]

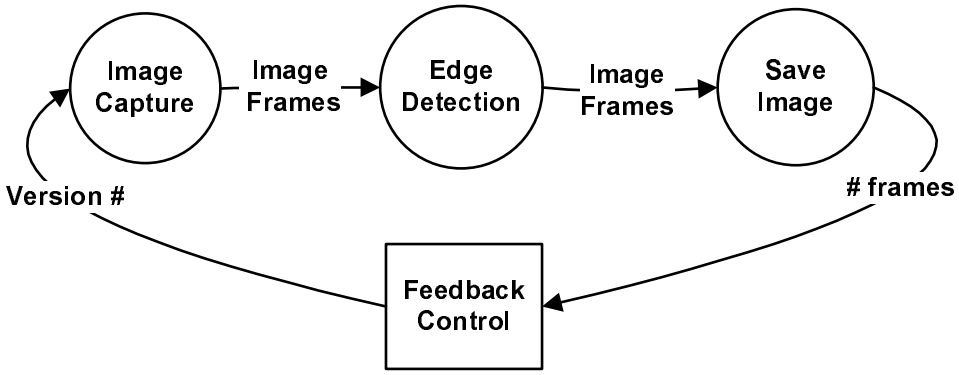


Figure 5.5 InfoSphere-target recognition application

frame from *Image Capture* to *Edge Detection* task.

Feedback control counts the frames during a two seconds window and compares the number of frames with a preset threshold. Once the frame rate drift from the threshold, the *Feedback control* triggers a fine-tuning loop that modifies the version's number sent to *Edge Detection* task. Version's number is modified upon receiving the trigger and the frame rate is watched within a smaller window to respond fast for any unwanted drift from the threshold. Version's number is increased if the frame rate is greater than the threshold, and decreased if the frame rate is less than the threshold, and hence the quality is increased or decreased, respectively.

It is not possible in InfoSphere platform to modify the system scheduling algorithm in order to have full control over resource allocation decisions. However, it is possible to allocate the operators in an application to specific hosts in the system using the tools provided by InfoSphere. The user-level allocation decisions are not as effective as a system-level allocation decisions that can access all the parameters in the system. The InfoSphere system was installed on four Virtual Machines (VMs) that are running Linux operating system. Then the sample

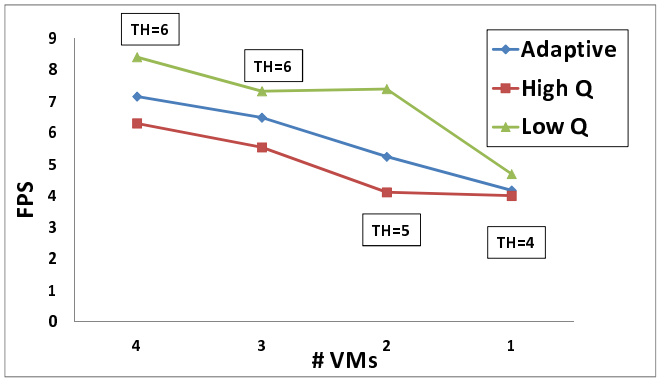
application in Fig. 5.5 is submitted to this system. After removing one VM at a time and fix the frame rate threshold accordingly, i.e., decreasing the threshold as the number of VMs is decreased, the experiment is repeated and averages of the frame rates and the qualities are taken after capturing 1000 frames for each number of VMs. Three algorithms run the same application; (i) High_Q which keeps the maximum quality of the frame image all the time, (ii) Low_Q keeps the quality of the frame image in its lowest quality so as to keep a higher frame rate, and (iii) Adaptive algorithm tries to keep the frame rate above the preset threshold by modifying the versions used and the quality of the result in consequence. Each operator in the application was allocated to specific host represented by one of the VMs in order to guarantee the same allocation decisions in the three algorithms.

Fig. 5.6 shows the results from running the described experiment on InfoSphere platform. Fig. 5.6(a) shows the frame rate (FPS) vs. the number of virtual machines (VMs). The FPS decreases for the three algorithms as the number of VMs decreases. FPS decreases, due to less computation capabilities available to the sample application as the number of VMs is decreased. Adaptive algorithm performs between both of the other algorithms and above the preset threshold shown on the figure.

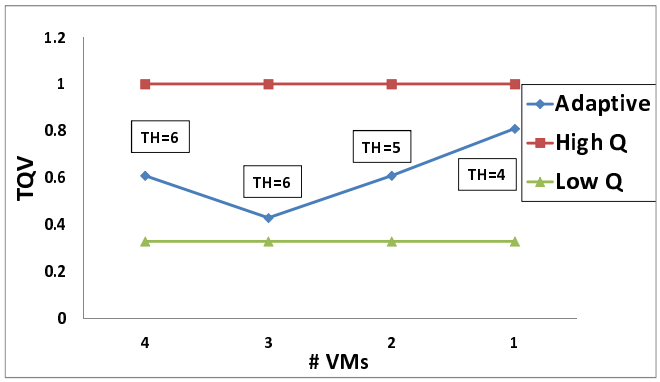
High_Q dropped below the threshold at two points when the number of VMs is three and four. High_Q algorithm runs highest quality versions of the application with a price of frame rate drop seen at these points.

On the other hand, Low_Q runs applications in low quality versions and hence light load on the VMs which results in higher FPS.

Fig. 5.6(b) shows the quality (TQV) vs. the number of virtual machines (VMs). Adaptive algorithm performs better than Low_Q and less than High_Q algorithm. There are no adaptation capabilities for High_Q and Low_Q algorithms; therefore their performance keeps the same value for all points in the figure. Adaptive algorithm tries to reduce the quality to keep the FPS above the threshold; therefore we see different quality on each point in the figure. When the number of VMs is three, Adaptive algorithm reduces the quality significantly to keep the load in the system within its computation capabilities and FPS above the threshold.



(a) FPS vs. number of VMs



(b) TQV vs. number of VMs

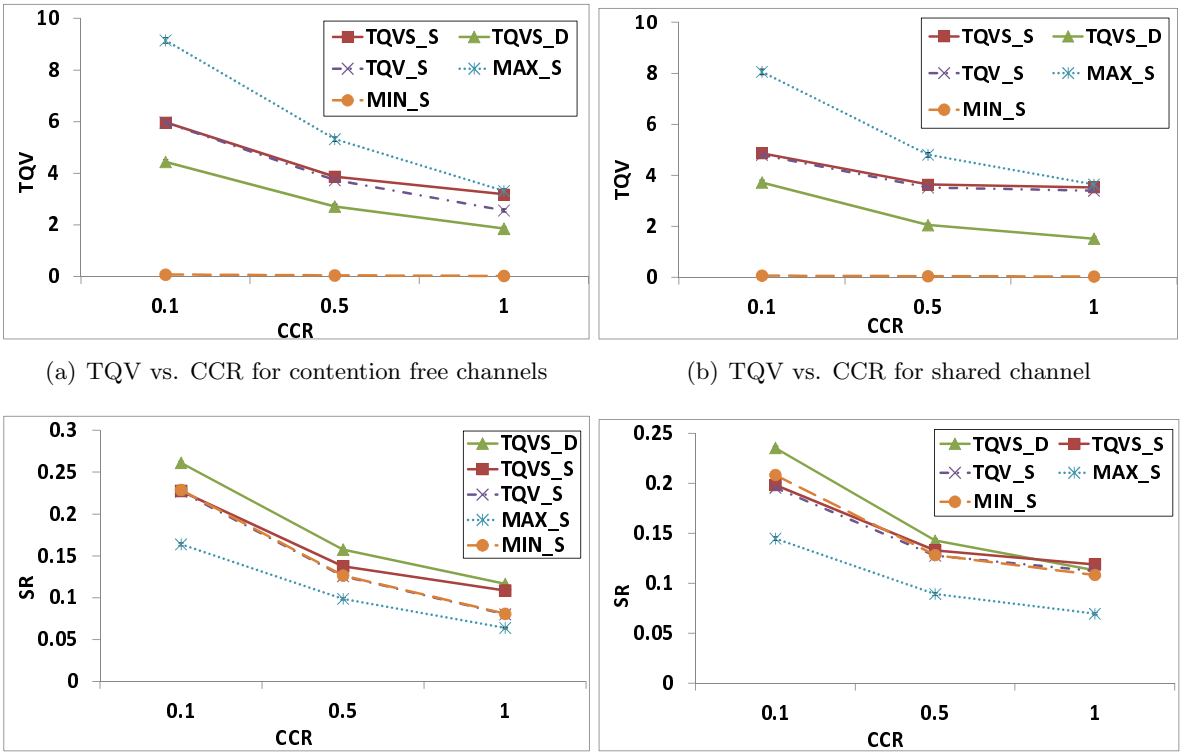
Figure 5.6 InfoSphere experiment results

5.7 Simulation Studies

To evaluate the proposed algorithms, a set of random DAGs that is an input to the algorithms were created. The same set of DAGs is also an input to three baseline algorithms that are used for performance comparison with the proposed algorithms. The baseline algorithms are also based on EDDF (Earliest Due Date First) policy:

- MIN_S uses lowest QoS level of the task during the assignment phase.
- TQV_S also uses lowest QoS level of the task during the assignment phase, but it has a TQV maximization phase.
- MAX_S considers tasks in their highest QoS levels. There is no TQV maximization in this algorithm as all the tasks are in their highest QoS levels.

Baseline algorithms do not consider security cost during the assignment phase.



(a) TQV vs. CCR for contention free channels

(b) TQV vs. CCR for shared channel

(c) SR vs. CCR for contention free channels

(d) SR vs. CCR for shared channel

Figure 5.7 Impact of CCR on TQV and SR for shared and contention free channels

The random generation of the DAGs is similar to that in [66]. The following parameters are used to generate each DAG used in our study.

- v: *Number of tasks per application.* The number of tasks per application is generated randomly from a uniform distribution with a mean equals to v.
- α: *The shape parameter of the DAG.* The number of levels in the DAG is randomly generated from a uniform distribution with mean value equals to $\alpha\sqrt{v}$. The number of tasks per level is randomly generated from a uniform distribution with mean value equals to $\sqrt{v}/\alpha$. The DAG will show more parallelism if $\alpha \ll 1.0$ and less parallelism if $\alpha \gg 1.0$. Therefore the DAG is balanced if $\alpha = 1.0$.

- β: *Sites homogeneity factor.* Each site has a computation factor that is generated randomly from a uniform distribution in the range $\{0.1 + ((1 - \beta)/|M|) * (m - 1), 0.1 + ((1 - \beta)/|sites|) * (m)\}$, where $m \in M$ is the number of site. As β decreases, the range of computation factor variation increases and vice versa.

- CCR: *Communication to computation ratio.* The application (DAG) is considered communication extensive if it has a high CCR and vice versa.

- *Sites: Number of sites.* The number of sites represents the maximum number of processors available to run the available applications.

- *numQoS: Number of QoS levels.* Number of QoS levels is generated for each task from a uniform distribution with an average equals to $numQoS$.

- The cost of byte processing by local and remote communication subsystem, and transfer over the network, TB.

- *avgcomp: The average computation time of the application.* The tasks computation times are generated from a uniform distribution with a mean equals to *avgcomp*. The average communication cost is calculated as $CCR * avgcomp$.

- *succNum: The number of successors.* Each task in the DAG has a number of successors that is generated from a uniform distribution with an average equals to $succNum$.

- SF: *Computation factor of the security per byte of the message.* SF is generated for each site by multiplying the computation factor of that site by 0.1.

- P: *Period of the application.* Each generated DAG has an end-to-end deadline that is the same as its period. P is randomly selected from the set {66.67, 50, 40, 33.33}, to simplify calculations of the LCM (Least Common Multiplier), which is equals to 200 in this case.

- γ: *The inter-arrival time of the applications.* The time between successive arrivals of the applications is drawn from an exponential distribution with a rate equals to γ.

The above input parameters were varied over the following:

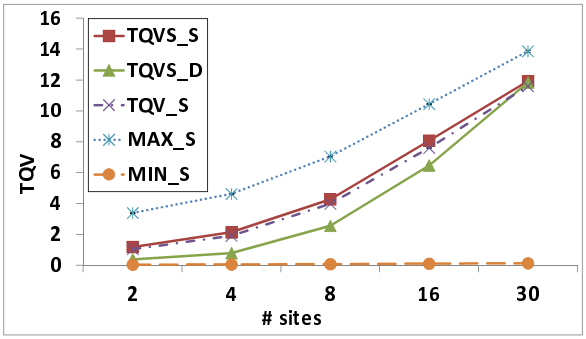
- $v = \{3, 4, 6, 8\}$
- $\alpha = \{0.5, 1.0\}$
- $\beta = \{0.1, 0.25, 0.5, 0.75, 1.0\}$
- $CCR = \{0.1, 0.5, 1.0\}$
- $Sites = \{2, 4, 8, 16, 30\}$
- $numQoS = \{5\}$
- $TB = \{0.01\}$
- $avgcomp = \{2\}$
- $succNum = \{3, 4\}$
- $\gamma = \{10, 50, 100, 200, 500\}$

Based on the above values, a total of (6,000) different DAGs are generated. Based on the parameter under study, most of the other combinations are used with the same average parameters and a random period. We allow each generated DAG to arrive to the system 192 times (except when studying CCR, it is 64 times) in series with selected inter-arrival time apart, and then the DAG departs after executing for LCM time of all the applications (200). The number of different DAGs used is dependent on the parameter under consideration.

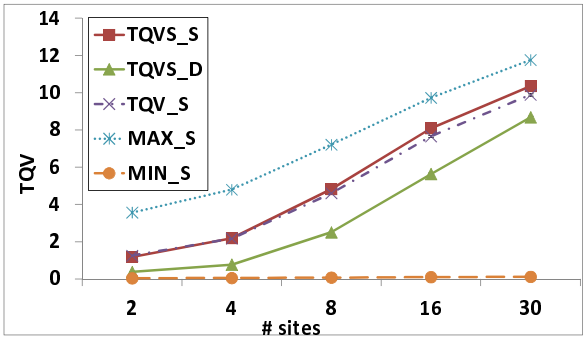
We studied the impacts of several parameters on the performance of the two proposed algorithms as compared to the baseline algorithms for contention free (fully connected sites) and shared channels. The parameters we studied are:

- CCR. We generated 16 different DAGs. Each one is allowed to arrive 64 times. We fixed the inter-arrival time to 10 time units. Thus we have a total of 1024 DAGs for each of other parameter then the average on all the parameters is taken. This is repeated 20 times.

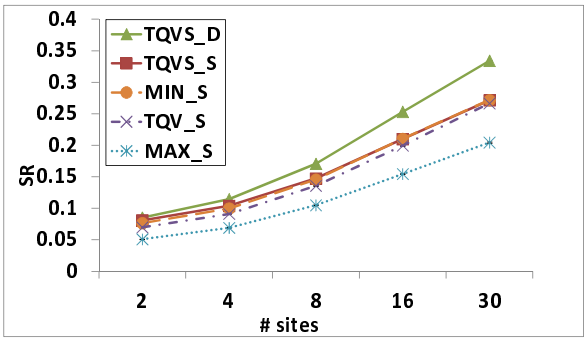
- β. The same in CCR, but the number of DAGs are 48 each is repeated for 192 times. Therefore the total is 9216 DAGs and the experiment is repeated 20 times.
- Number of sites. The same as β.
- γ. The same as β but the inter-arrival time, γ, is varying over its range.



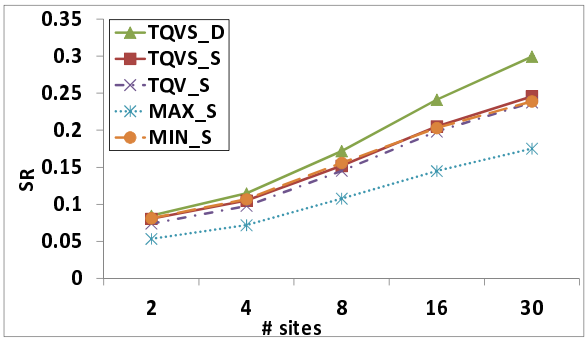
(a) TQV vs. number of sites for contention free channels



(b) TQV vs. number of sites for shared channel



(c) SR vs. number of sites for contention free channels



(d) SR vs. number of sites for shared channel

Figure 5.8 Impact of number of sites on TQV and SR for shared and contention free channels

In all of the simulation experiments, we assumed the security levels of all tasks are constant and equal to RL, which is assumed to be 1. Performance investigation under dynamic variation in RL is reserved as a future work.

Fig. 5.7, 5.8, 5.9 and 5.10 show our study results based on the simulation experiments for different parameters.

The performance trend of all the algorithms is the same for shared and contention free channels. Performance magnitudes for contention free channels are slightly larger than the

performance of shared channel due to the cost of data transmitting between tasks incurred in shared channel case.

In all simulation studies, MAX_S algorithm has the lower performance in SR and the highest in TQV, which is due to the nature of assignment phase of this algorithm. This algorithm deals with tasks in their highest QoS levels which results in a lower number of admitted tasks, but a higher TQV as compared to other algorithm.

MIN_S has the lowest performance in TQV since tasks are admitted based on lowest QoS levels and stay on these levels, hence the resulted TQV is the lowest.

TQVS_D has the highest performance in SR during almost all experiments. This is due to the freedom of this algorithm during the assignment phase, where even the previously assigned tasks can be considered again for a new assignment each time a new application arrives.

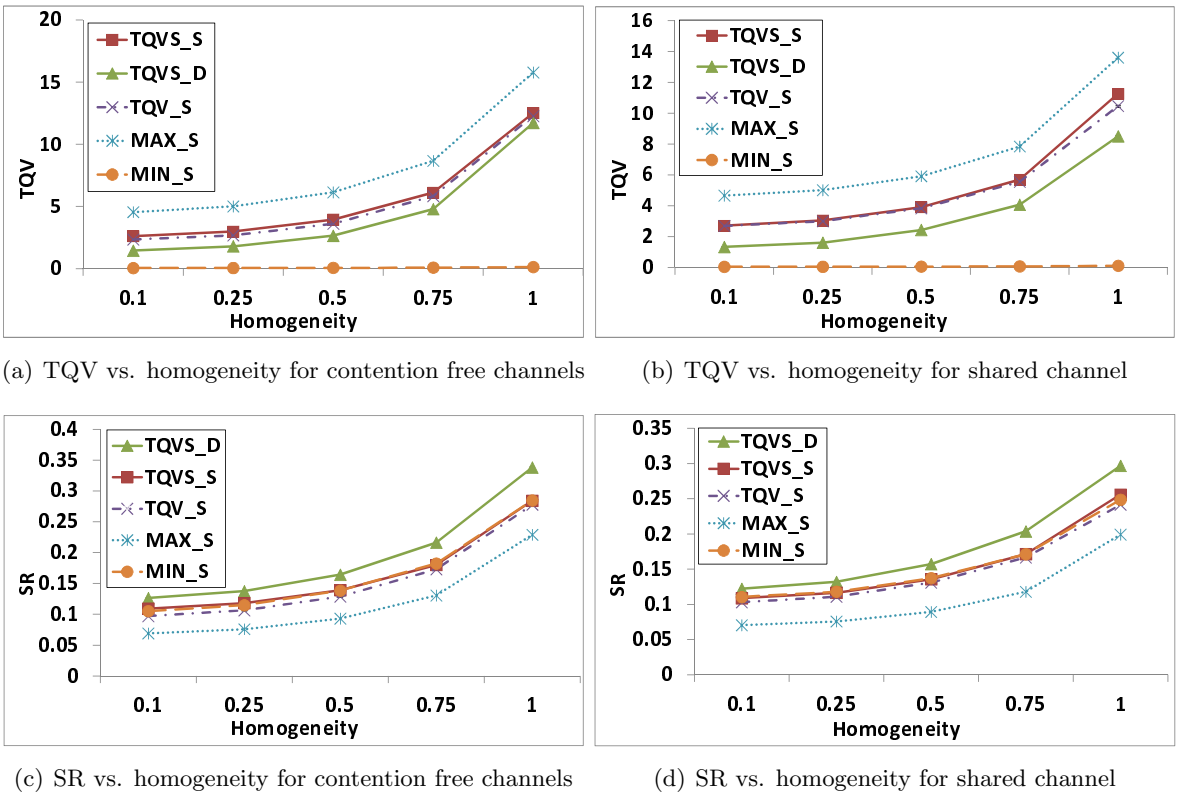
5.7.1 Impact of CCR

Fig. 5.7 shows the proposed algorithms performance in terms of SR and TQV as compared to the baseline algorithms when communication to computation ratio (CCR) varies from 0.1 to 1. Both of the proposed algorithms outperform baseline algorithms in terms of SR, see Fig. 5.7(d) and 5.7(c). SR decreases as the exchanged data sizes between tasks in arrived applications increases. The relationship between TQV and SR when the resources are scares tends to be reversal. Therefore the TQVS_D in Fig. 5.7(b) and 5.7(a) tends to be inferior of the others in terms of TQV while TQVS_S performs better than other algorithms except MAX_S where tasks are in their highest QoS levels.

It is shown for single channel network, see Fig. 5.7(d), when CCR equals one that TQVS_S outperforms TQV_D in terms of SR. This behavior is due to the fact that as the data size increases the cost of securing the sent/received data becomes significantly high that affects the assignment decision of the tasks. Hence in TQVS_S the cost of security (data encryption/decryption cost) is already accounted for in the assigned tasks (especially the tasks of second and beyond instances of the application), therefore assignment decision of arrived tasks will be more accurate in this case as compared to TQVS_D that deals with tasks without

completely considering security cost.

5.7.2 Impact of Number of Sites



(a) TQV vs. homogeneity for contention free channels

(b) TQV vs. homogeneity for shared channel

(c) SR vs. homogeneity for contention free channels

(d) SR vs. homogeneity for shared channel

Figure 5.9 Impact of sites computation homogeneity on TQV and SR for shared and contention free channels

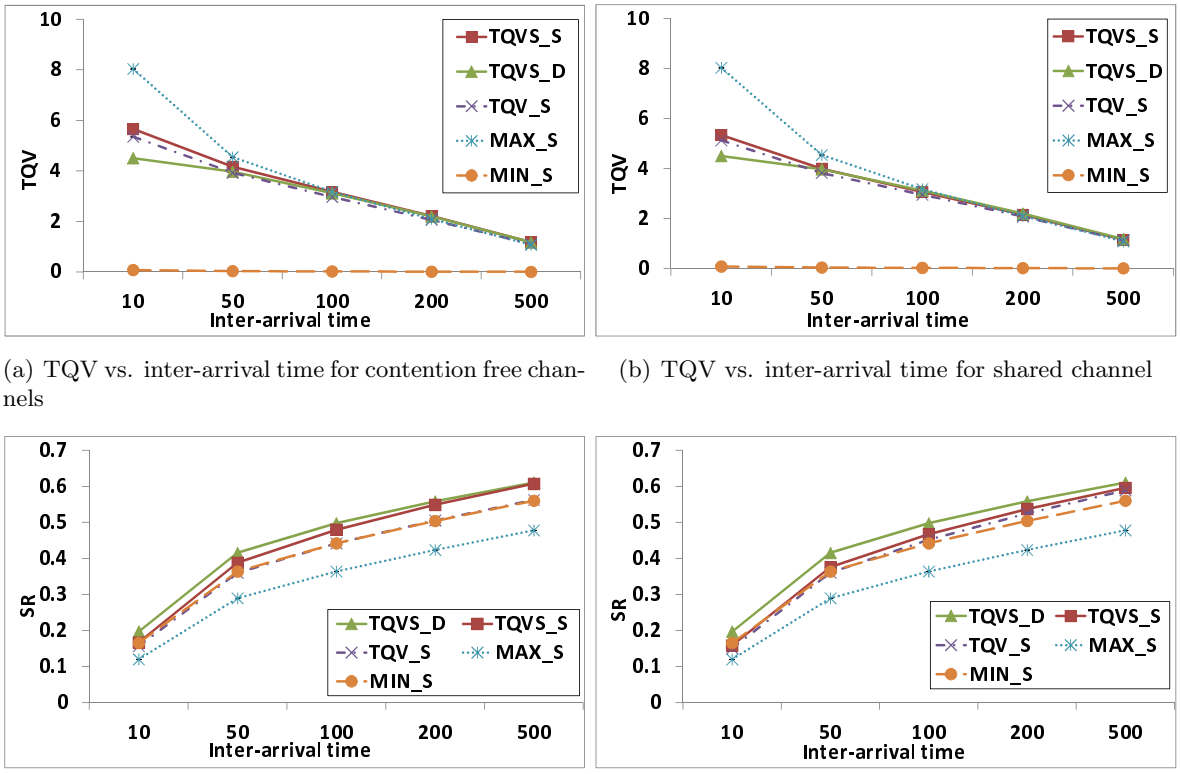
Fig. 5.8 shows the proposed algorithms performance in terms of SR and TQV when the number of sites is varied from 2 to 30. The general trend shows that as the number of sites increases the magnitude of SR and TQV increases for contention free and shared channels networks. This behavior is expected since the computational capabilities increase as the number of sites increases.

In Fig. 5.8(d) and 5.8(c), TQVS_S performs the same as MIN_S in terms of SR and a little bit higher than TQV_S, because TQVS_S uses lowest QoS level of the task during admission and assignment which is the same as MIN_S algorithm. TQV_S algorithm uses the same assignment method as MIN_S, however the task to be assigned will not find the same ready

times of the sites as in MIN_S due to the already running tasks. This and the additive security cost make the application more susceptible to miss its deadline and hence to be rejected and explains the difference in SR.

In Fig. 5.8(a) when the number of sites increases TQVS_S, TQVS_D and TQV_S become closer and closer to each other due the availability of slack to raise QoS levels of the admitted applications to the best values. While this is not the case for MAX_S because it is already has the lowest SR and tasks are in their highest QoS levels. In Fig. 5.8(b) although algorithms are getting closer to each other but not as in contention free case due to the cost needed in transmitting data in a shared channel.

5.7.3 Impact of Sites Computation-Homogeneity-Factor



(a) TQV vs. inter-arrival time for contention free channels

(b) TQV vs. inter-arrival time for shared channel

(c) SR vs. inter-arrival time for contention free channels

(d) SR vs. inter-arrival time for shared channel

Figure 5.10 Impact of applications inter-arrival times on TQV and SR for shared and contention free channels

Fig. 5.9 shows the performance of TQVS_S and TQVS_D in terms of SR and TQV as

compared to baseline algorithms for shared and contention-free channels. From the figure it is clear that as the sites become more homogeneous in computation capabilities the performance magnitude increases.

In Fig. 5.9(c) and 5.9(d) TQVS_S and MIN_S have the same performance due to admission and assignment method that considers tasks in lowest QoS levels and better than TQV_S for the same reason given in Subsection 5.7.2.

In Fig. 5.9(a) the performance of TQVS_S, TQV_S and TQVS_D is getting closer as the slack of increasing QoS levels becomes larger as sites becomes more homogeneous. This is not the case for shared channel because we still have the bottleneck of communications between tasks due to single communication channel, see Fig. 5.9(b).

5.7.4 Impact of Inter-arrival Time

Fig. 5.10 shows our algorithms performance in terms of SR and TQV as compared to baseline algorithms for shared and contention-free channels when the inter-arrival time is varied from 10 to 500. Fig. 5.10(c) and 5.10(d) show that as the inter-arrival time between applications increases SR increases. In Fig. 5.10(c) TQVS_S and TQVS_D are getting closer as the inter-arrival increases because almost when each application arrives it finds the system empty hence both algorithms behaves the same. For MIN_S and TQV_S the performance is the same starting from 50, where the two algorithms find almost the same conditions of the sites load. MIN_S and TQV_S perform bellow TQVS_S and TQVS_D.

In Fig. 5.10(d) TQVS_S, TQVS_D, TQV_S and MIN_S are getting closer despite that TQV_S and MIN_S assignment method does not take security into consideration and this is because the main factor here is the communication cost that is constrained by the shared channel availability. MAX_S still performs less than all the algorithms because it uses the highest QoS levels of the tasks during assignment which makes rejection highly likely to occur during the first phases due to communication tightness and to sites overloading.

Fig. 5.10(a) and 5.10(b) show that as inter-arrival time increases TQV decreases, because the number of tasks in the system in time unit decreases. As the inter-arrival time increases

all algorithms, except MIN_S, tends to have the same performance because all of them find almost the same slack in the sites to increase their QoS levels to an amount close to MAX_S.

In summary TQVS_D algorithm performs better than other algorithms in almost all the simulation experiment. To use this algorithm one should take into consideration that this algorithm might incur more cost to move tasks from site to site depending on the type of tasks under consideration. TQVS_S performs below TQVS_D and sometimes the same as some of the baseline algorithms, however it can be used in all cases where there is no need to move tasks from sites after assignment, which eliminates the tasks movement cost.

5.8 Conclusions

In this chapter, the problem of dynamic assignment-and-scheduling of security and QoS-aware application on heterogeneous distributed real-time systems is addressed. Two heuristic algorithms to solve the problem for shared and contention free communication channels are proposed. The proposed algorithms are evaluated by extensive simulation experiments using wide range of randomly generated workload. The evaluation showed that the proposed algorithms outperformed the baseline algorithms in all cases.

CHAPTER 6 Conclusions and Future Work

6.1 Conclusions

Secure exchange of data in security sensitive real-time systems has to be guaranteed in a timely fashion. The existence of QoS (accuracy) and security-aware applications, that are based on and leveraged from the imprecise computation paradigm, creates a tradeoff between these requirements on one hand and accuracy on the other hand. Types of QoS and security-aware applications range from simple independent tasks to a task graph modeled as Directed Acyclic Graph (DAG) with dependency and precedence relationships. The primary focus of this research has been on scheduling of the tasks in a given system (uniprocessor or distributed system) to maximize system schedulability while maintaining acceptable QoS. In this regard, the scheduling goal for uniprocessor system is to maximize security and QoS of the system. In heterogeneous distributed real-time system the goal of scheduling is to maximize the QoS while satisfying the system constraints. Our main contributions can be succinctly stated as follows:

(1) Uniprocessor dynamic scheduling problem of independent tasks with QoS and security requirements to maximize the combined QoS and security level was studied.

- The problem was formulated as a MILP and proved to be NP-hard.
- A heuristic scheduling algorithm (SQV_EDF) to maximize SQV (combined Security and QoS Value) was proposed.
- Evaluations using an extensive simulation experiments showed that SQV_EDF outperformed a set of baseline algorithms. For a special case where tasks are preempt-

able and have the same arrival times, SQV_EDF performance was close to optimal solution.

(2) Static allocation of task graphs modeled as Directed Acyclic Graphs (DAGs) with QoS and security requirements in heterogeneous distributed system problem was addressed and studied.

- The problem was formulated as a MINLP.
- A centralized heuristic algorithm (EDD_LQL) for static allocation with the goal of maximizing TQV of the system was proposed.
- The algorithm was evaluated using extensive simulation studies for shared and contention free communication channels. Evaluations showed that EDD_LQL algorithm outperformed a set of baseline algorithms in most of the cases.

(3) Dynamic allocation of task graphs modeled as Directed Acyclic Graphs (DAGs) with QoS and security requirements in heterogeneous distributed system problem was addressed and studied.

- The problem was formulated and formally stated.
- As the problem is NP-hard, two heuristic algorithms were proposed to solve this problem; TQVS_S and TQVS_D.
- Evaluation of the proposed algorithms using an extensive simulation experiments showed that the proposed algorithms outperformed a set of baseline algorithms.

(4) An experimental evaluation to validate the simulation results for some of the concepts in this research was given.

- InfoSphere platform (by IBM Corporation) was used to validate the effectiveness of some of the concepts used in this dissertation. InfoSphere middleware was installed on a set of virtual machines.
- A sample application that runs edge detection algorithm on captured frames and then save the frames to the local disk was implemented.

- An adaptive algorithm was designed to keep the frame rate of the images above a threshold utilizing the QoS-awareness of the application.
- Comparison of the adaptive algorithm with other algorithms that do not adapt to fluctuations in system load showed the effectiveness of the adaptive algorithm to keep the system frame rate above the threshold. This was achieved by degrading QoS of some of the frames when it is needed.

6.2 Future Work

The proposed work in this dissertation does open up several directions for future research. Future research can take any of several dimensions used in the proposed work, i.e., workload nature, system type (real-time or non-real-time), system design objective and constraints that should be met in the system.

- In system objective dimension, relaxing the constraint of meeting a hard deadline and consider minimizing the tardiness of the system, opens up a direct extension to this research.
- Applying the concept of imprecise computation for tasks in general purpose systems can improve performance guarantees for real-time applications including QoS and data security.
- Integration of multiple security properties such as confidentiality, integrity, and availability into system model in this research introduces a fine tuning capability to the performance of the whole system.
- For distributed system part, security can also be optimized along with QoS. This might be achieved by giving a new formulation of the problem and introducing new performance metrics.
- Defining new metrics that consider variations of data sizes according to QoS levels that directly affects the communication cost produces a new dimension for this research.

- Integration of communication channels characteristics into this work produce a deep refinement of the channel abstract level assumed throughout this work and hence help developers to produce applications with more system awareness and considerations.
- Other goals and/or constraints such as fault tolerance and energy consumption of the system can be included along with security and QoS considered in this work, which produce an interesting optimization problems.
- Real-world implementation of the algorithms developed in this research in a fully controllable system (installed on virtual machines and its scheduler can be modified) will provide more insight into the solutions and validate the simulation results.

Bibliography

[1] D.T. Cole, P. Thompson, A.H. Göktoğan, and S. Sukkarieh. System development and demonstration of a cooperative uav team for mapping and tracking. *International Journal of Robotics Research*, 29(11):1371–1399, 2010.

[2] System S Stream Computing at IBM Research (Last accessed June 21, 2012. http://public.dhe.ibm.com/software/data/sw-library/ii/whitepaper/SystemS_2008-1001.pdf.

[3] C.S.R. Murthy and G. Manimaran. *Resource management in real-time systems and networks*. The MIT Press, 2001.

[4] J.A. Stankovic. *Deadline scheduling for real-time systems: EDF and related algorithms*. Springer, 1998.

[5] R. Frederick and V. Jacobson. Rtp: A transport protocol for real-time applications. *IETF RFC3550*, 2003.

[6] G. Wettstein and MS Johannes Grosen. Gaining the middleground: a linux-based open source middleware initiative. In *Proceedings of the 4th annual Linux Showcase & Conference-Volume 4*, pages 47–47. USENIX Association, 2000.

[7] F. Krichen, B. Zalila, M. Jmaiel, and B. Hamid. A middleware for reconfigurable distributed real-time embedded systems. *Software Engineering Research, Management and Applications 2012*, pages 81–96, 2012.

[8] F. Kon, M. Román, P. Liu, J. Mao, T. Yamane, L. Magalhaes, and R. Campbell. Monitoring, security, and dynamic configuration with the dynamictao reflective orb. In *Middleware 2000*, pages 121–143. Springer, 2000.

[9] D.C. Schmidt and C. Cleeland. Applying patterns to develop extensible orb middleware. *Communications Magazine, IEEE*, 37(4):54–63, 1999.

[10] J. Balasubramanian, S. Tambe, C. Lu, A. Gokhale, C. Gill, and D.C. Schmidt. Adaptive failover for real-time middleware with passive replication. In *Real-Time and Embedded Technology and Applications Symposium, 2009. RTAS 2009. 15th IEEE*, pages 118–127. IEEE, 2009.

[11] V. Subramonian, G. Deng, C. Gill, J. Balasubramanian, L.J. Shen, W. Otte, D.C. Schmidt, A. Gokhale, and N. Wang. The design and performance of component middleware for qos-enabled deployment and configuration of dre systems. *Journal of Systems and Software*, 80(5):668–677, 2007.

[12] Real-Time CORBA specification (Last accessed June 23, 2012). http://www.ois.com/images/stories/ois/real-time corba specification 05-01-04 jan 2005.pdf.

[13] J. Balasubramanian, B. Natarajan, D. Schmidt, A. Gokhale, J. Parsons, and G. Deng. Middleware support for dynamic component updating. *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pages 978–996, 2005.

[14] B. Zalila, L. Pautet, and J. Hugues. Towards automatic middleware generation. In *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*, pages 221–228. IEEE, 2008.

[15] V. Van Tan, D.S. Yoo, and M.J. Yi. Security in automation and control systems based on OPC techniques. In *Strategic Technology, 2007. IFOST 2007. International Forum on*, pages 136–140. IEEE, 2009.

[16] S. Chaumette, R. Laplace, C. Mazel, and A. Godin. Secure cooperative ad hoc applications within UAV fleets position paper. In *Military Communications Conference, 2009. MILCOM 2009. IEEE*. IEEE, 2010.

[17] K.D. Kang and S.H. Son. Systematic Security and Timeliness Tradeoffs in Real-Time Embedded Systems. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE Computer Society Washington, DC, 2006.

[18] K.D. Kang and S.H. Son. Towards security and QoS optimization in real-time embedded systems. *ACM SIGBED Review*, 3(1), 2006.

[19] T. Xie and X. Qin. Improving security for periodic tasks in embedded systems through scheduling. *ACM Trans. Embedded Computing Systems*, 2007.

[20] C. Ekelin. Clairvoyant non-preemptive EDF scheduling. In *18th Euromicro Conference on Real-Time Systems, 2006.*

[21] K.J. Lin and et al. *Imprecise Results: Utilizing Partial Computations in Real-time Systems*. National Aeronautics and Space Administration, 1987.

[22] J.W.S. Liu, K.J. Lin, W.K. Shih, A.C.S. Yu, J.Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *Computer*, 24(5):58–68, 1991.

[23] T.D. Braun, H.J. Siegel, and A.A. Maciejewski. Static mapping heuristics for tasks with dependencies, priorities, deadlines, and multiple versions in heterogeneous environments. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 78–85. IEEE, 2002.

[24] R. Gonçalves, R.S. de Oliveira, and C. Montez. Design pattern for the adaptive scheduling of real-time tasks with multiple versions in rtsj. In *Chilean Computer Science Society, 2005. SCCC 2005. 25th International Conference of the*, pages 9–pp. IEEE, 2005.

[25] T.D. Braun, H.J. Siegel, A.A. Maciejewski, and Y. Hong. Static resource allocation for heterogeneous computing environments with tasks having dependencies, priorities, deadlines, and multiple versions. *Journal of Parallel and Distributed Computing*, 68(11):1504–1516, 2008.

[26] X. Zhu, X. Qin, and M. Qiu. Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters. *Computers, IEEE Transactions on*, (99):1–1, 2011.

[27] R. Jejurikar. Energy aware non-preemptive scheduling for hard real-time systems. In *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*, pages 21–30. IEEE, 2005.

[28] W. Li, K. Kavi, and R. Akl. A non-preemptive scheduling algorithm for soft real-time systems. *Computers & Electrical Engineering*, 33(1):12–29, 2007.

[29] R. Al-Omari, A.K. Somani, and G. Manimaran. An adaptive scheme for fault-tolerant scheduling of soft real-time tasks in multiprocessor systems. *Journal of Parallel and Distributed Computing*, 65(5):595–608, 2005.

[30] T. Xie, A. Sung, and X. Qin. Dynamic Task Scheduling with Security Awareness in Real-Time Systems. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 274a–274a, 2005.

[31] M. Lin, L. Xu, L.T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu. Static Security Optimization for Real Time Systems. *IEEE Transactions on Industrial Informatics*, 5(1), 2009.

[32] T. Abdelzaher and KG Shin. End-host architecture for QoS-adaptive communication. In *Fourth IEEE Real-Time Technology and Applications Symposium, 1998. Proceedings*, pages 121–130, 1998.

[33] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek. A resource allocation model for QoS management. In *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*, pages 298–307. IEEE, 2002.

[34] J.P. Hansen, J.P. Lehoczky, and R. Rajkumar. Optimization of quality of service in dynamic systems. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 95. IEEE Computer Society, 2001.

[35] S. Ghosh, R.R. Rajkumar, J. Hansen, and J. Lehoczky. Scalable resource allocation for multi-processor qos optimization. 2003.

[36] S. Ghosh, J. Hansen, R. Rajkumar, and J. Lehoczky. Adaptive QoS optimizations with applications to radar tracking. In *10th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*. Citeseer, 2004.

[37] S. Ronngren and B.A. Shirazi. Static multiprocessor scheduling of periodic real-time tasks with precedence constraints and communication costs. In *hicss*, page 143. Published by the IEEE Computer Society, 1995.

[38] K. Ramamritham. Allocation and scheduling of precedence-related periodic tasks. *IEEE Transactions on Parallel and Distributed Systems*, 6(4):412–420, 1995.

[39] G.L. Stavrinides and H.D. Karatza. Scheduling multiple task graphs with end-to-end deadlines in distributed real-time systems utilizing imprecise computations. *Journal of Systems and Software*, 83(6), 2010.

[40] G.U. Srikanth, AP Shanthi, V.U. Maheswari, and A. Siromoney. A survey on real time task scheduling. *European Journal of Scientific Research*, 69(1):33–41, 2012.

[41] Y.K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys (CSUR)*, 31(4):406–471, 1999.

[42] N. Arora. Analysis and performance comparison of algorithms for scheduling directed task graphs to parallel processors. *ANALYSIS*, 4(2), 2012.

[43] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[44] E. Demeulemeester and W. Herroelen. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management science*, pages 1803–1818, 1992.

[45] T. Javidi and D. Teneketzis. An approach to connection admission control in single-hop multiservice wireless networks with qos requirements. *Vehicular Technology, IEEE Transactions on*, 52(4):1110–1124, 2003.

[46] R van Slyke and Y Young. Finite horizon stochastic knapsacks with applications to yield management. *OPERATIONS RESEARCH*, 48(1):155–172, 2000.

[47] K. Jeffay, D.F. Stanat, and C.U. Martel. On non-preemptive scheduling of periodic and sporadic tasks. In *Proceedings of the Twelfth IEEE Real-Time Systems Symposium*, pages 129–139. Citeseer, 1991.

[48] R. Nassiffe, E. Camponogara, and G. Lima. Optimizing quality of service in real-time systems under energy constraints. *ACM SIGOPS Operating Systems Review*, 46(1):82–92, 2012.

[49] L. Niu. Energy efficient scheduling for real-time embedded systems with qos guarantee. *Real-Time Systems*, 47(2):75–108, 2011.

[50] K.J. Lin, S. Natarajan, and J.W.S. Liu. Imprecise results: Utilizing partial computations in real-time systems. In *Real-Time System Symposium*. Citeseer, 1987.

[51] S. Misailovic, S. Sidiroglou, H. Hoffmann, and M. Rinard. Quality of service profiling. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1, 2010*, pages 25–34. ACM, 2010.

[52] M. Bohlouli and M. Analoui. Grid-HPA: Predicting Resource Requirements of a Job in the Grid Computing Environment. *International Journal of Computer Science*, 32, 2008.

[53] David Pisinger. *Algorithms for Knapsack problems.* PhD thesis, Dept. of Computer Scince, University of Copenhagen, Universitetsparken 1, DK-2100 Copenhagen, Denmark, February 1995.

[54] L. Georges, P. Muhlethaler, and N. Rivierre. A few results on non-preemptive real time scheduling. *RAPPORT DE RECHERCHE-INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE*, 2000.

[55] R. Gopalakrishnan and G.M. Parulkar. Bringing real-time scheduling theory and practice closer for multimedia computing. In *Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 1–12. ACM New York, NY, USA, 1996.

[56] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change protocols for priority-driven preemptive scheduling. *Real-Time Systems*, 1(3):243–264, 1989.

[57] S.K. Dhall and CL Liu. On a real-time scheduling problem. *Operations Research*, pages 127–140, 1978.

[58] IBM ILOG CPLEX Mathematical Optimization Technology (Last accessed June 21, 2012). http://www-01.ibm.com/software/integration/optimization/cplex-cp-optimizer/.

[59] K. Ramamritham and J.A. Stankovic. Scheduling algorithms and operating systems support for real-time systems. *Proceedings of the IEEE*, 82(1):55–67, 1994.

[60] X. Zhu, J. Zhu, M. Ma, and D. Qiu. SAQA: A Self-Adaptive QoS-Aware Scheduling Algorithm for Real-Time Tasks on Heterogeneous Clusters. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 224–232. IEEE, 2010.

[61] G. Lima, E. Camponogara, and A.C. Sokolonski. Dynamic reconfiguration for adaptive multiversion real-time systems. In *Euromicro Conference on Real-Time Systems*, pages 115–124. IEEE, 2008.

[62] G.L. Stavrinides and H.D. Karatza. Scheduling real-time dags in heterogeneous clusters by combining imprecise computations and bin packing techniques for the exploitation of schedule holes. *Future Generation Computer Systems*, 2012.

[63] J.P. Hansen, S. Ghosh, R. Rajkumar, and J. Lehoczky. Resource management of highly configurable tasks. 2004.

[64] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness. A Series of Books in the Mathematical Sciences.* WH Freeman and Company, San Francisco, Calif, 1979.

[65] D.M. Batista, N.L.S. da Fonseca, and F.K. Miyazawa. A set of schedulers for grid networks. In *Proceedings of the 2007 ACM symposium on Applied computing*, page 213. ACM, 2007.

[66] H. Topcuoglu, S. Hariri, and M.Y. Wu. Task scheduling algorithms for heterogeneous processors. In *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pages 3–14. IEEE, 2002.

[67] N. Al-Oudat and M. Govindarasu. Qos and security aware allocation of directed acyclic graph on heterogeneous distributed real-time systems. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 50–56. IEEE, 2012.

[68] J. Zhu, X. Zhu, and J. Jiang. Improving adaptivity and fairness of processing real-time tasks with qos requirements on clusters through dynamic scheduling. *Information Processing Letters*, 2011.

[69] L. Khan and W. Fan. Tutorial: Data stream mining and its applications. In *Database Systems for Advanced Applications*, pages 328–329. Springer, 2012.

[70] IBM InfoSphere Streams Information Center (Last accessed June 23, 2012). http://publib.boulder.ibm.com/infocenter/streams/v2r0/index.jsp.

[71] IBM InfoSphere Platform (Last accessed June 21, 2012). http://www-01.ibm.com/software/data/infosphere/.