Minimizing the Makespan of Diagnostic Multi-Query Graphs in Embedded Real Time Systems

DISSERTATION

To Obtain the Degree of Doctor of Engineering

Submitted By

Nadra Tabassam

Submitted To

Department of Electrical Engineering and Computer Science

Chair of Embedded System

University of Siegen

Siegen 2020

Supervisor And First Appraiser

Prof. Dr.Roman Obermaisser

University of Siegen

Second Appraiser

Prof. Dr. Ali Jannesari

Iowa State University

Date of the Oral Examination

27th January 2020

I would like to dedicate my work to my beloved mother who was always there through my thick and thin.

ACKNOWLEDGEMENTS

I would like to thank my supervisor ”Professor Roman Obermaisser” for his feedback, cooperation and guidence. In addition, I would like to express my gratitude to the staff of Embedded System specially Manuela Popp and Stefan Otterbach. I would like to thank my friends Maryam, Sara, Adele and Sitara for accepting nothing less than excellence from me. Last but not least, I would like to thank my family for supporting me spiritually throughout my PhD and my life in general.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

In recent years numerous control systems were deployed with safety-critical components comprised of real time embedded systems. These systems have electronic control units connected by networks, sensors and actuators. These systems demand high levels of reliability and have strict timing constraints especially in case of fault occurrence. One method to achieve this reliability is to introduce continuous monitoring and active diagnosis in the system. For implementing the active diagnosis in real-time systems, processing of diagnostic queries needs to satisfy the strict timing bounds. Our optimization algorithms minimize the overall makespan of Diagnostic Mutli-Query Graphs (DMG) in order to meet timing bounds. The overall makespan of the DMG is minimized by applying query optimization and graph optimization techniques. Query optimization is applied to the Fault Diagnostic Queries (FDQs) that are the part of each node of the DMG. The best access methods and best query execution plans are selected for the optimization of FDQs. The graph optimization is also applied to our DMG without affecting the semantics of FDQs. The proposed techniques are tested with different types of parameters including (i). FDQs, (ii). Left Deep Tree (LDT) and Bushy Tree (BT) and (iii) network topologies. The proposed techniques are tested in the context of two domains including (i) vehicles and (ii) Heating Ventilation and Air Conditioning System (HVAC). These proposed methodologies show a significant reduction of the makespan and give convincing results with different parameters.

Zusammenfassung

In den letzten Jahren wurden zahlreiche Steuerungssysteme entwickelt. Eingesetzt mit sicherheitskritischen Komponenten, die aus eingebetteten Echtzeitsystemen bestehen. Diese Systeme verfügen über elektronische Steuergeräte, die über Netzwerke, Sensoren und Aktoren miteinander verbunden sind. Diese Systeme erfordern ein hohes Maß an Zuverlässigkeit und unterliegen insbesondere im Fehlerfall strengen Zeitvorgaben. Eine Methode, um diese Zuverlässigkeit zu erreichen, ist die Einführung einer kontinuierlichen Überwachung und aktiven Diagnose im System. Für die Implementierung der aktiven Diagnose in Echtzeitsysteme muss die Verarbeitung von Diagnoseabfragen die strengen zeitlichen Grenzen erfüllen. Unsere Optimierungsalgorithmen minimieren die Gesamtreichweite von Diagnostic Mutli-Query Graphs (DMG), um die Timing-Grenzen einzuhalten. Die Gesamtdauer des DMG wird durch den Einsatz von Techniken zur Abfrageoptimierung und Diagrammoptimierung minimiert. Die Query-Optimierung wird auf die Fault Diagnostic Queries (FDQs) angewendet, die Teil jedes Knotens des DMG sind. Für die Optimierung von FDQs werden die besten Zugriffsmethoden und besten Query-Ausführungspläne ausgewählt. Die Graphenoptimierung wird auch auf unser DMG angewendet, ohne die Semantik von FDQs zu beeinflussen. Die vorgeschlagenen Techniken werden mit verschiedenen Arten von Parametern getestet, darunter (i). FDQs, (ii). Linker Deep Tree (LDT) und Bushy Tree (BT) und (iii) Netzwerktopologien. Die vorgeschlagenen Techniken werden im Rahmen von zwei Bereichen getestet, darunter (i) Fahrzeuge und (ii) Heizungs-, Lüftungs- und Klimaanlagen (HVAC). Diese vorgeschlagenen Methoden zeigen eine signifikante Reduzierung der Markenbreite und liefern überzeugende Ergebnisse mit verschiedenen Parametern.

CHAPTER 1
INTRODUCTION

This chapter elaborates the background, motivation, objectives and summary of our contribution. The framework that is designed for the realization of our active diagnosis technique is also introduced briefly.

1.1 Background

Safety-critical systems are different from conventional systems. Failures of these systems may result in one or more of the following outcomes: (i). loss of life, (ii). damage of property, (iii). environmental harm. In recent years numerous control systems were deployed with safety critical components comprised of real-time embedded systems. These systems have electronic control units connected by networks, sensors and actuators. These systems demand high levels of reliability and have strict timing constraints, especially in case of fault occurrence. One method to achieve this reliability is to introduce continuous monitoring and active diagnosis in the system. An important step in fault treatment is a diagnosis, which determines the cause of a failure in terms of localization and nature. Using a root cause analysis, anomalous behaviors and states are traced back to the originating fault. Depending upon on how diagnostic information is used, one can distinguish between the passive and active diagnosis. Passive diagnosis stores the diagnostic information and uses it later for finding the fault in the system. On the other hand, in case of active diagnosis, both the detection and diagnosis of the fault is performed online within the strict timing defined by the system. For implementing active diagnosis in real-time systems, processing of diagnostic fault queries needs to satisfy strict timing bounds.

Real-time systems based on active diagnosis are different from other common systems in terms of their timing requirement for diagnosis. In the context of applications having

an active diagnosis of faults, each diagnostic task has to complete within a pre-defined deadline. If a real time application is unable to fulfill the requirement of fault diagnosis at run time within hard time bounds, then ultimate consequences can be fatal. In order to maintain the timeliness of these applications, it is imperative to be guaranteed that all the tasks complete within the pre-defined deadline. Example systems which benefit from active diagnosis are health management systems [1], command/control systems [2], electric power distribution applications [3] and aircraft control systems [4]. Flight control, autonomous driving, space crafts and military surveillance applications are some examples of these systems. For example in case of autonomous vehicles, one goal is to achieve safe and autonomous driving at affordable cost [5]. Therefore one needs to consider the fact that autonomous vehicles are fail operational and in case of high automation levels there is no driver to take control of the vehicle. Similarly in Unmanned Aerial Vehicles (UAVs), fault monitoring and diagnosis are implemented to ensure reliability and safety [6].

Another example is a Fault Tolerant Control System (FTCS). An active FTCS system reacts to the malfunctioning of system components by re-configuring the controller based on the information received from a Fault Detection and Diagnosis (FDD) unit. The major objectives of an active FTCS is to develop an effective FDD scheme to provide information about the fault with minimal uncertainties in a timely manner [7]. Therefore the fault detection based on-line diagnosis improves the overall reliability of these safety critical systems. The active diagnosis in these systems is imperative because these applications have timing constraints for fault recovery (e.g., actuator freezing, limited time until correct service is required again or to prepare for the next fault when combining diagnosis with conventional replication) [8].

There are mainly two solutions in state of the art to ensure the reliability of these systems. The first solution is based on adding redundant components in the system. But it is often not feasible as this increases the overall cost of the final product [9]. Another solution is the continuous monitoring of the system for temporary or permanent component failures.

There are two types of diagnostic techniques: (i). passive diagnosis that stores the diagnostic information and analyzes it later for fault detection and repair and (ii). active diagnosis that analyzes the diagnostic information at run time so that an immediate recovery action can be taken [10]. Embedded systems are typically comprised of numerous actuators and sensors. All these components are subject to faults which can result in the deviation of their values from the correct range. These faults may cause damage including loss of life and harm to the environment. These process abnormalities have a serious impact as the industry can lose hundreds of billions of dollars [11][12]. Active diagnosis helps to detect the faults earlier and prevents this damage of system and avert great losses [13].

In order to improve the safety, reliability and performance, numerous researchers have worked on the problem of fault detection. Therefore different adequate and effective methods have been designed in this context [14]. The fault detection techniques are classified into two types (i). history-based methods and (ii). model- based methods [15]. History based methods are dependent on the historical data processing and are further subdivided into qualitative (rule-based) and quantitative (statistical) methods [16]. Another aspect of these applications is the processing of high volumes of data. There is often a huge amount of data transmitted among the different control units in real-time systems. It is important to store and process the real time data reliably and timely, otherwise the system can face a catastrophic situation. There are many real-time database management systems with the features of concurrency control and transaction scheduling. Real-time database management systems that are commercially available include, Pervasive.SQL [17], Polyhedra [18], TimesTen [19] and Berkeley DB [20]. Research projects that are designed by using these database systems include DeeDs [21] and BeeHive [22].

As a summary, there are two important aspects of real time systems. One aspect is the reliability, and another aspect is the huge amount of data processing between the different control units. Reliability can be achieved by introducing active diagnosis in the system. And for timely data processing and analysis, one can employ the real-time database man-

agement system. Hence it is clear from the above discussion that time is the most important entity in hard real time systems. All the processes, including active diagnosis and the huge amount of data processing, should be completed within the pre-defined deadline of the system. Therefore one needs to determine for each computational activity the Worst-Case Execution Time (WCET). WCET analysis can be defined as the determination of upper bounds of execution times. Knowledge about the WCET is a vital step for the development and validation of real-time systems. The determination of WCETs is important because hard real-time systems should satisfy the strict timing constraints expected from the system. In general upper bounds on the execution times are required to show the satisfaction of these constraints.

1.2 Framework for Active Diagnosis

Fig.1.1 gives an overview of the proposed framework. Our fault detection and diagnosis platform is based on fault diagnostic queries that use rule-based inference and semantic web technology to identify faults in the system. In such an approach, a diagnostic knowledge base (DKB) describes the structure of the Open Distributed Real Time Embedded Systems (ODRE) using semantic web technology, i.e. constituting components and their interfaces, defines faulty or anomalous behaviour, rules for the identification of faults and the respective recovery actions for mitigating failures. A directed graph of diagnostic rules known as Diagnostic Multi-Query Graph (DMG) is the central element of this approach. The diagnostic inference process will be temporally and spatially decomposed by introducing inference steps called symptoms. These symptoms and diagnostic features will be stored in real-time databases, part of which will be timely and consistently replicated to enable the distributed execution of rules. Each rule (i.e. symptom, fault, action) will be realized as a query on the diagnostic facts within a real-time database, while edges in the DMG specify input/output relationships via the real-time database.

The scope of this thesis is to minimize the Makespan (total schedule length) of the

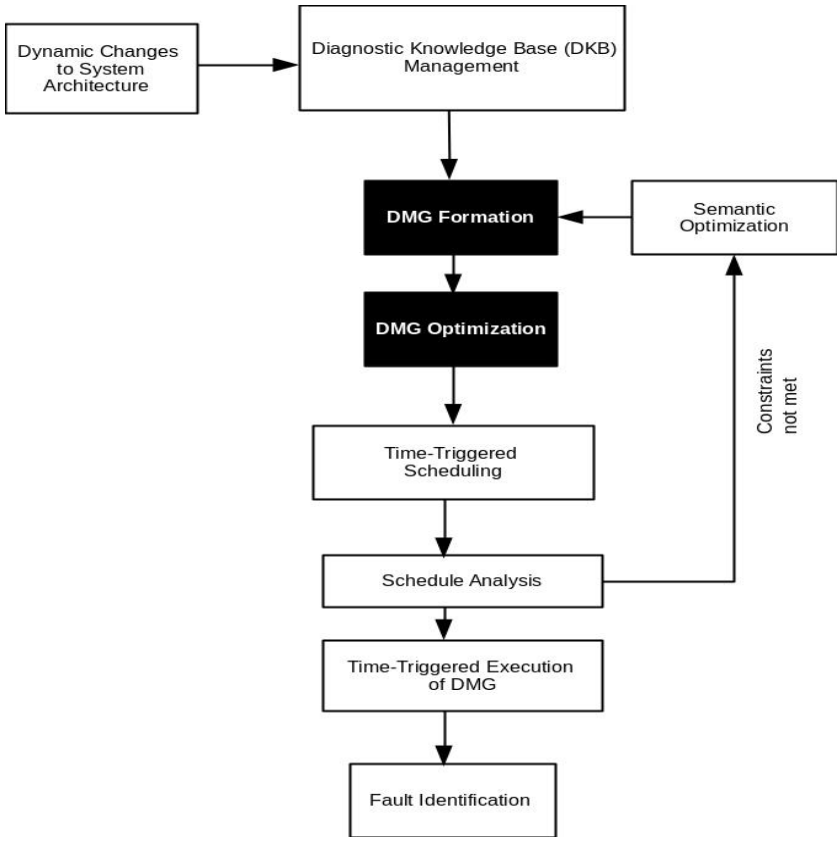


Figure 1.1: Frame Work for Active Diagnosis

DMG. Therefore, Fault Diagnostic Queries (FDQs) are optimized in a manner that their WCET is minimized. Minimization of WCET of FDQs minimizes the overall makespan of the DMG. This work is important in the context of real-time systems as minimization of makespans of DMGs enables the fault diagnosis process to be completed in stringent timing deadlines provided by the system. We have marked the blocks in the black that are covered in this thesis (Fig. 1.1).

1.3 Motivation

In recent years, the application areas of real-time embedded systems have evolved to include stringent timing constraints, reliability requirements, and the need for an open world assumption, i.e. components are integrated at run time to realize emerging global services dynamically. These systems demand high reliability, cost efficiency and support for stringent timing constraints. Although the active diagnosis may benefit dependability of the Open Distributed Real-time Embedded (ODRE) systems significantly but the present techniques do not consider the dynamic nature of the system and the strict real time requirements collectively. Moreover, for fault tolerant systems to work efficiently, the active diagnosis must be reliable itself to ensure that a fault affecting the diagnostic mechanisms does not cause a wrong fault recovery action. These aspects are usually not considered in the present day techniques.

In the state-of-the art, a wide variety of methods of active diagnosis is available. An example of active diagnosis is the classical PMC model [23] ,which is extended to deal with distributed analysis and transient faults in [24]. Online diagnosis algorithms for run time isolation and recovery are introduced in [25]. Fault detection and identification using Bayesian inference and dynamic decision networks with parameterized fault trees are employed for an on-board architecture for active diagnosis in [26]. Another approach for active online diagnosis uses residuals which are generated by parameter estimation, taking into account the parametric peturbations in a non-linear model [27]. None of the existing

diagnostic techniques satisfies the combined support for openness, real-time and reliability as required by the ODRE systems. Either the diagnostic models are rigid without the support of dynamically changed system structure or the computational cost is unsuited for ODRE systems.

1.4 Objectives of Thesis

The principal objective of our work is to implement techniques beyond the state of the art, through which the process of fault detection and diagnosis can be optimized. More precisely, we have proposed methods based on active diagnosis that satisfy the reliability and real-time requirements. Our proposed technique enables the system to complete the process of fault detection and diagnosis within a predictable time. For this purpose, we have taken into account the methodologies through which we can minimize the makespans of FDQs. These FDQs are used for active diagnosis in time-driven and non-preemptive real-time systems. In order to complete these FDQs in the stringent timing deadline provided by the system, different optimization techniques are applied. These optimization techniques are applied to minimize the overall WCET of FDQs. Therefore the minimization of WCET helps the process of fault detection to be completed in the pre-defined deadline of the system.

1.4.1 Summary of Main Contributions

This research focuses on devising techniques through which the process of fault detection and diagnosis can be completed with real-time guarantees. The important contributions of this thesis are as follows:

- FDQs are created for fault detection and diagnosis. These FDQs are based on the features and symptoms extracted directly from the sensor data. Each FDQ is written in SQL format and executed over the database that contains the values from the sensors data.

- Based on FDQs a Diagnostic Multi-Query Graph (DMG) is designed. Each DMG is comprised of multiple FDQs. Each node of the DMG contains an FDQ while each edge has a weight which is equivalent to the amount of data that is transferred between parent and child node.

- In order to minimize the overall WCET of the FDQs different optimization techniques are applied. These techniques enable the fault diagnosis process to complete within the stringent timing deadlines provided by the system. Our main focus of the optimization is based on the two aspects: (i). optimization of FDQs and (ii). optimization of the DMG.

- Optimization of FDQs: This is actually a multi-query optimization performed for the transformation of the DMG. Different steps, including query parsing and selection of the most optimal query execution plans, have the highest impact on the time of query execution. These multi-query optimization methods are comprised of (i). selection of best join orders, (ii). selection of best Query Execution Plan (QEP) and (iii). selection of the best access methods for a specific FDQ.

- Optimization of DMG: This technique is based on the graph pruning and merging of graph nodes. An optimized DMG with a lower number of processing nodes and an optimized QEP are more likely to be scheduleable and detect the faults within the given deadline. Bigger DMGs with large numbers of fault queries are more difficult to schedule as there is a risk that these graphs cannot complete their execution within the predefined deadline of the system. It is important to optimize the DMG in a manner that it may determine the fault in the system within the timing constraints defined by the system. Our proposed optimization technique satisfies the reliability and real-time requirements and enables the process of fault diagnosis to be completed within a predictable time.

1.5 Structure of Thesis

The structure of the thesis is as follows:

- Chapter 2 introduces the fundamental definitions related to this work.
- Chapter 3 covers the already existing approaches (related work) that are present in the state of art.
- Chapter 4 explains the architecture and system model, which is proposed in this work.
- Chapter 5, 6, 7, 8 and 9 gives a detailed overview of the techniques which are used for the implementation of this work. Examples are also incorporated in each chapter along, with experimental results for evaluation.
- Chapter 10 concludes the thesis with an executive summary.

CHAPTER 2
FUNDAMENTAL CONCEPTS

The research scope of real-time system comprises different aspects, from top-level design issues down to the implementation of individual components. In order to further clarify the motivation and the following chapters of the thesis, the fundamental definitions are described in detail.

2.1 Real Time Systems

Real-time refers to quantitative measurements of time. In other words, the time in this context is measured with a physical clock. So whenever the time is quantified by using a physical clock, it is actually the measurement of real-time. An example use of this time quantification is observed in a chemical plant. When a power plant will attain a pre-determined temperature of 260°C, the system will automatically shut down the heater within the time interval of 25 milliseconds. In this scenario, the time is quantified by using a physical clock present in the chemical plant. Therefore a real-time system is the one in which time is used as a quantitative notion. Any system which is not using quantitative time is not considered as a real-time system.

A hard real-time system should process the information and responds within the specific time bounds provided by the system. This implies that all results of a particular task should come before a defined deadline, otherwise the system can face a catastrophic situation. Real-time systems are not only dependent on the results of transactions but also the instant of time at which these results are produced. Real-time systems can be hard or soft in terms of meeting their deadlines [28]. Typical examples of real time systems include command control systems, autonomous driving systems and air traffic control system. In the case of soft real-time systems missing deadline affects the utility of the system, but there are

no dangerous consequences. There are numerous examples in which applications require real-time computing. Some of them are elaborated in the next section. Table 2.1 shows the properties of hard and soft real time systems [29].

Table 2.1: Hard Vs Soft Real-Time System Properties

Characteristics	Hard Real-Time	Soft Real-Time
Response time	hard deadline	soft deadline
Safety	critical	non-critical
Size of files	small	large
Load in peak performance	predictable	degraded
Data Integrity	long term	short term

2.1.1 Examples of Real Time Systems

Automotive applications: Control system in automotive determine the speed of the car with the help a cruise control system. This system also monitors the fuel consumption, mileage and average speed of the car.

Metal industry applications: These systems are typically used in controlling processes such as casting, hot rolling, cold rolling, finishing, soaking, annealing and other functions related to metal processing.

Aerospace: In modern-day aircraft, there is an option of ”autopilot”. As soon as this option is selected, a computer system takes control of all the tasks including navigation, takeoff and landing of the aircraft. This computer obtains the X, Y, Z positions of the aircraft and compares them with the already specified data. If the direct of aircraft is changed (start moving towards the wrong destination), then this computer will detect the fault and take the corrective measures. This process is highly sensitive in terms of time as all the corrective actions should be taken within a few milliseconds.

Based on these examples of real-time applications, it is observed that the timing constraints are not a peculiar property of the design of these systems, but they are determined by the environment in which these control systems are operating [30].

2.1.2 Properties of Real-Time Systems

This section highlights the properties of real-time systems. These characteristics distinguish real-time systems from conventional systems. It is not necessary that each real-time system exhibits all of these properties. The properties depend on the context of the application.

Timing Constraints: Every real-time system has a time constraint. One aspect of this time constraint is known as the deadline of a task. A task deadline specifies the time before which it has to be completed and produce its results. A Real-Time Operating System (RTOS) ensures that all the tasks meet their respective deadlines.

Correctness Criteria: In the case of real-time systems, correctness is not only related to the accuracy of results, but it also implies the time at which these results are generated. The results produced after the deadline are considered to be incorrect.

Safety-Critical: Safety-critical systems are simply systems with safety requirements. Safety is defined as the reliability with respect to critical failure modes. This means that safety is a special type of reliability.

Concurrency Control: Real-time systems are often complex and involves a large amount of data communication among different sensors and actuators. Therefore, real-time systems should process the data from different sensors and actuators concurrently. The loss of sensor data may cause a system failure.

Task Criticality: Task criticality is the measure of the cost of the failure of a task. The high critical tasks require higher levels of reliability. Task criticality is an important aspect and should be taken into account when fault tolerance is introduced in the system.

Meeting Deadlines: During overload situations, the system should meet the deadlines of critical tasks as missing the deadline may cause a catastrophic situation.

Reliability and safety: Reliability is defined as the probability that a system will perform its expected tasks for a specified period of time under certain constraints provided by the environment. Safety is the reliability with respect to critical failure modes. Both of

these objectives can be achieved by introducing active diagnosis in real-time systems [31].

Predictability: Predictability in real-time systems is defined in different ways. In the case of static real-time systems, the overall performance of the system can be analyzed deterministically at development time. In dynamic real-time systems, a stochastic analysis is often required for the performance evaluation. Predictability of these systems in particular implies that the timing requirements of critical tasks are guaranteed in all loads and fault conditions of the system [32].

2.2 Worst-Case Execution Time (WCET)

The problem of finding the WCET for real-time applications is difficult because of various reasons. There is a large gap between the access times of main memory and cycle times of modern processors. There are numerous methods in the state of the art for measuring the WCET of a particular task. Two important aspects should be considered for determining the WCET. These aspects include (i). safety and (ii). precision. A modular approach for the timing analysis splits a task into sub-tasks where, these sub-tasks deals with different properties including control flow, sequential flow of instructions and execution times of various instructions. Hard real-time systems have stringent timing requirements. Each task in the system has to satisfy the strict timing bound provided by the system. Generally, an upper bound for each task is calculated to ensure that the timing constraints given by the system are not violated. However, in case of programs, it is not always possible to derive the upper bounds on the execution times. A guarantee about the WCET of the program can only be given if the worst-case input of the program is known. We assume that the real-time system is comprised of a number of tasks. Each task has varying execution times depending upon the types of inputs. The shortest execution time taken by a task is known as the Best Case Execution Time (BCET), while the longest time is known as the WCET [33]. There are two types of methods for the WCET calculation of the task.

- Static Methods: These methods do not rely on real hardware. These methods take

the code and analyze the possible control flow paths (CFPs). Later these CFPs are combined with abstractions of the hardware model, and the upper bound of the task is obtained. One such method is elaborated in [34]. Static methods compute an upper bound on the tasks and guarantee that the task will complete its execution within the time bound provided by the system.

- Measurement-Based Methods: These methods executes the task on a simulator or the actual hardware with specific inputs and derive the maximum and minimum time of a task. By observing the maximum execution time of the task during many executions with different inputs, an approximation of the WCET is obtained.

2.3 Time Triggered Systems

The tasks within Time Triggered (TT) systems follow a pre-computed schedule. Each task has its own allocated time slot during which it is executed. It can be assumed that these systems are predictable in nature as the system designer can predict in advance how a system will behave at a certain instant of time [35]. Meeting all the timing constraints and knowing details about the system behaviour can be considered as predictability [36]. However, this type of model is often difficult to implement in real world, and different approximations are being widely used in practice. The most accurate approximation of this model is comprised of periodic tasks running in a non-preemptive manner [37]. TT systems with these kinds of properties have a high level of timing predictability along with low jitter [38].

2.4 Distributed Systems

A distributed system is a collection of autonomous elements with computing power and appears as a single entity for the end-user. Considering this definition provides the two basic aspects of distributed systems: (i) the computing element is normally referred to as

a node and can behave autonomously. (ii) For end users, all the nodes behave like a single system which means that all the nodes should collaborate in order to make the system work. In practice, these nodes work together in order to achieve a single goal, by exchanging messages. Therefore the nodes communicate using a network. Distribution transparency is an important design objective. It is also inevitable that any node of the system may fail and the processes running on that node may stop. These aspects of distributed systems should be kept in mind in design [39].

2.5 Databases

A database systematically stores similar data in a manner that the data retrieval becomes fast. Therefore it provides an organized mechanism for storing, managing and retrieving the information. Databases are comprised of multiple tables, which are known as relations. Each column is an attribute while each row is a record in the relation. A database is considered to be more efficient as compared to a flat file systems. A database management system maintains the data properties like integrity and confidentiality. ANSI has defined a three level architecture for a database which is comprised of (i) the internal level, (ii) the conceptual level and (iii) the external level [40].

1. The internal level determines where data is stored at the disk. It contains all the details related to the number of bytes and how these bytes are transferred from one storage device to others.

2. The conceptual level describes the details about the logical view of the data, including queries.

3. The external level describes the details about the interaction of users with the application programs.

2.5.1 Database Model

The database model defines the details about the logical model of the data. There are three types of database models: (i). hierarchical models, (ii) network models and (iii) relational models.

Hierarchical Model

In this model, data is stored in a hierarchical manner. Every entity has only one parent but can have numerous children. There is one entity at the top of this hierarchy which is known as root.

Network Model

In this model, the entities are stored in the form of a graph, and there are certain entities which are accessible through different paths.

Relational Model

In this model, the data is stored in a two dimensional structure known as relations. These relations are connected to each other by primary and foreign keys [41].

2.5.2 Transaction

A database transaction is a task that is performed over the records stored in a relation. These transactions may retrieve, update or delete the record. In a relational database, each transaction should maintain the ACID properties (Atomicity, Consistency, Isolation, and Durability). The atomicity means that a transaction is considered as one atomic unit. The transaction consists of multiple statements, and all the statements should be considered as one unit. If one of the statements in a transaction is not completed, then the whole transaction should rollback.

Consistency means that the database should remain in a consistent state after the transaction

is completed. Any data written to the database should be valid according to the defined rules including triggers, cascades and constraints. The property of durability ensures that once a transaction is committed, it should remain committed even if the system fails or restarts. Completed transactions are written to non-volatile memory. Isolation is relevant when there are multiple parallel executions of transactions in a database. It should be ensured that each transaction is executed independently of the other transactions [42].

2.5.3 Real Time Database Management System

Data management in a real-time system is application dependent. As the complexity of real-time system increases, it becomes mandatory to manage and store the data in an organized and systematic manner. Real-time database management systems (RTDBMS) are different from the conventional DBMS. RTDBMS retrieve and store the data in a similar manner, but there is an additional feature of timing constraints. Each transaction has to complete within the pre-defined deadline of the system. Example applications that require the processing of large amounts of data along with stringent timing requirements include health care systems, flight control systems, radar tracking and autonomous vehicles.

A conventional DBMS only focuses on the fast average response and the high throughput for transaction executions. In contrast, RTDBMS are evaluated on the basis of the fact of how often a transaction misses its deadline. Traditional databases deal with persistent data-sets. The goal of query processing is to achieve a low average response time and high throughput. However, RTDBMS deals with time-sensitive data (i.e. data that is out-dated after a certain period of time). One of the first real-time database implementations was the disk-based transaction processing test bed known as RT-CARAT [43]. Other projects are REACH (Real-time Active and Heterogeneous mediator system project) [44] and STRIP (Stanford Real-time Information Processor) [45]. Examples of commercial RTDBMS are eXtremeDB [46], Eaglespeed-RTDBMS [47] and SolidDB [48]. All of these RTDBMS focus on memory management and memory optimization in order to achieve real-time per-

formance.

2.5.4 Properties of Real Time Database Management Systems (RTDBMS)

There are various important properties of RTDBMS which should be considered before using them. Some of them are highlighted in this section.

Transaction, Data and System Characteristics: RTDBMS should maintain temporal consistency, logical consistency and timing properties of data and transactions.

Transaction Processing: The major issue in transaction processing is predictability [49]. If real time transactions miss their deadlines the end result can be catastrophic. Therefore, it is mandatory for the system to predict that these transactions are completed before their pre-defined deadlines. This prediction is only possible if the WCET of the transaction is already known. In the case of RTDBMS, there are numerous potential sources for unpredictability [50]:

- Conflicts between data and resources.
- Dynamic paging and interference at input/output interfaces.
- Abortion of transactions and resultant rollbacks.

Dynamic paging and I/O conflicts can be resolved by using main memory databases. Therefore, priority oriented I/O controllers can be used to solve the problem of I/O unpredictability.

Scheduling Real Time Transactions: The Scheduling policy for transactions describes how priorities are assigned to individual transactions for their execution. There are numerous transaction policies defined in the state of the art. A transaction that is scheduled in a RTDB is known as a task [51]. Scheduling in this context is comprised of allocating tasks to the processors so that they can be completed on time. A typical real-time system is comprised of multiple tasks that should be executed concurrently. Each task computes certain values which can be used by the other tasks. Also, each task has a deadline before

which the task should be completed [52]. Transactions can be categorized as hard, soft and firm. These terms describe the value given to a certain transaction when it has to meet its deadlines [53].

Hard deadline transactions are time sensitive. If the transaction deadline is missed, then there can be a huge loss. These types of transactions are normally part of safety-critical systems [30]. A large negative value is assigned to a transaction after the deadline is missed. Soft deadline transactions in real-time systems can slow down their response time in case of high processing loads. This means that soft real-time transactions do not have stringent deadlines. For example, flight reservation systems are flexible in processing the task and also have large database files. Typically the value given to the system by a transaction is not equal zero after the deadline is passed. Firm transactions do not dissipate any value to the system after the deadline is passed.

2.6 Query Optimization

The domain of query optimization encompasses extensive prior work. There are several contexts and different angles of query optimization covered by researchers. In this section, we will discuss the structure of a query optimizer and different methods to implement the concept of optimization. Query optimization is comprised of two stages: (i) rewriting and (ii) planning [54].

2.6.1 Rewriting

Rewriting transforms the query and establishes a more efficient query as compared to the original one. In this phase, rewriting only considers the static properties of the query and does not consider the actual query cost. Because of the nature of this step, rewriting operates only at a declarative level [55].

2.6.2 Planning

Planning is the most important module of query optimization. It analyzes all the execution plans and selects the one with the minimum cost. This step uses a certain search strategy which identifies the best query execution plan. The search space of execution plans is determined by two other modules present in the optimizer known as the algebraic module and the method module.

Algebraic Module: This module determines the order of algebraic expressions for the planner. Trees based on the query operations are created. These trees are represented in the form of algebraic expressions. There are numerous trees that can be created by considering different orders of query operations. Therefore each tree has different cost in terms of resource, time and memory utilization.

Method Module: After getting the tree (as a set of actions) from the algebraic module, this module produces the complete Query Execution Plan (QEP) to specify the join operations and indices that are used [56].

2.6.3 Cost Model

This module generates the arithmetic formulas that are employed to find the cost of the QEP. For all access methods and join types, there is a formula that gives the cost. These formulas are based on the utilization of different factors like disk and CPU utilization, buffer management and I/O processing. The most important input factors for the formula are the size of relations and the buffer pool used by each operation.

2.6.4 Size Distribution Estimator

This module determines the size of relations and results of each sub-query. This module uses these estimations for finding the cost of the complete QEP [57].

CHAPTER 3

RELATED WORK

This chapter is divided into four sections. The first section describes the related work in the domain of query optimization. The second section describes the related work in the domain of fault tree models. The third section elaborates the graph based optimization techniques. The forth section describes the fault detection techniques that are using graph based techniques. The fifth section elaborates the related work in the domain of Real-Time Database Management systems (RTDMS). Sixth and the last section describes the related work in the domain of WCET analysis.

3.1 Query Optimization

There is a large number of problems that can be taken into account when considering the query performance. These problems fall under two different categories: (i) database normalization and (ii) un-optimized queries having no indexes. There are multiple techniques for query optimization, which work in collaboration with each other. These optimization techniques may include indexing and selection of the best join order. By applying these techniques, the overall performance of the query is optimized. The query optimization technique described in [58] optimizes the query by breaking down the complex queries into smaller chunks and then execute all of them in parallel.

Optimization solutions select the (i) best QEP and (ii) the most efficient query. Both of these optimization techniques help in minimizing the overall execution time of queries. Query optimization is dependent on the database management system and the format of query [59]. dQUOB (Dynamic Query Objects) is a system that is dynamic, and it enables the end-users to create a query explicitly for the data that is required. These specific queries acquire that data in a fashion that is much more pragmatic for the end-user. The dQUOB

system has been applied to two systems, namely autonomous robotics and software visualization for atmospheric data. The dQUOB system provides a relational data model along with the SQL based query access to the streaming data [60]. The process of fault handling is added to the Continuous Event Processing (CEP) Queries. These queries are specifically designed for the CEP systems. These CEP queries are designed in a customized manner by the user so that in case of fault occurrence, only the CEP related to a particular fault is halted [61].

An algorithm was conceptualized and implemented for the optimization of multi-join queries for smart grid data in a database [62]. This technique uses a genetic algorithm comprised of the following steps: (i) encoding of virtual connections, (ii) selection of the individual solution, (iii) application of crossover operation, (iv) mutation and (v) termination. The proposed solution improves the overall convergence rate and accuracy of the selected solutions. Query optimization techniques in multi-tenant databases are different as compared to other databases. In multi-tenant databases, every customer is only restricted to its schema for the usage of the database. When a data retrieval query is written by a particular tenant, the syntax of the query is checked. Results in this context show that these optimization techniques provide efficient and cost-effective queries for data retrieval [63]. Query optimization techniques employed in embedded database systems play an imperative role during the design of an embedded system. During the first phase of this process, the RTDBMS is selected, as the selection of the database engine plays a vital role. The selection of efficient RTDBMS enables the embedded system to work correctly and efficiently. During the second phase of this process, the query optimization algorithms are implemented. Authors in [8] used the particle swarm based query optimization technique. A high inertia weight is employed in finding a new search space. After the execution of a maximum number of iterations, a final inertia weight is obtained and the inertia weight is decreased with different values of particles.

Multifarious re-writing approaches for cost-based optimization of SQL queries are con-

ferred in [64]. The exploratory investigation presented in this context served as a tuning tool to enhance the query processing performed for production databases. Experimental evaluation based on realistic scenarios shows that query performance and operational costs are improved. A swarm intelligence (called Bees Algorithm) method for the Multi-Join Query Optimization (MJQO) problem is presented in [65]. The proposed method is capable of finding an efficient solution with a fastest convergence rate as compared to the already existing solutions. It minimizes the response time of query processing. Simulation results show that this algorithm solves the MJQO problem in a shorter time as compared to the accustomed particle swarm optimization (PSO) techniques.

Genetic Algorithm (GA) based query optimization is widely used in literature [66]. Implementation of GA based optimization techniques provides efficient results when compared with conventional optimization techniques. A GA solves the problem of join ordering effectively in the context of large join queries. There is a huge amount of data processing among the different control units of an aerospace management system. One important problem of this system is the writing of optimized data extraction queries. However, the design of join queries solves this problem. A Balanced Compromise between Objective Space and Parameter Space (BCOP) algorithm based on GA for multi-join queries provides an effective model for solving this problem. The proposed technique [67] takes the features of Bee Colony (BC) and Simulated Annealing (SA). BC and SA resolve the problems of rapid convergence and local optima . Query optimization in distributed database systems requires both global and local optimizations. Global optimization is more critical as compared to local optimization. As in the case of global optimization, data is processed and shared at multiple sites [68][69].

An essential goal of a query processor is to produce efficient query results in the shortest time. One method of achieving this goal is to generate QEPs with minimum processing costs. The Query Optimizer (QO) is a substantial component of a query processor. The QO always selects the best QEP from multiple QEPs. QEPs are comprised of different

access methods and join orders. Best QEP is the one which takes the minimum processing time [70]. The research on query optimization of heterogeneous databases is an important area. A heterogeneous database results in many problems including (i) data communication and (ii) timeliness. A hibernate middleware based technology provides the solution for heterogeneous database systems [71].This technology solves different problems including (i) query decomposition, (ii) data conversion, (iii) data types, (iv) query optimization, and (v) query translation. Minimization of response time by using hibernate middle-ware technology provides better results in the context of heterogeneous databases .

Query processing that is performed on a central node incurs more communication overhead due to frequent exchange of data between the sink and the sensor nodes. This type of query processing is normally implemented in distributed systems. The proposed Distributed Nested Loop Join Processing (DNLJP) scheme groups sensors based on geographic locations to form a cluster. The query processing is performed in a distributed manner over the data collected from different geographic locations. DNLJP also optimizes the execution time of a query [72]. Caching methods are used for both single and multi-query optimization. The cost of evaluation can be reduced by exploiting common sub-expressions and maintaining an intermediate data structure. Single query optimization is integrated with the cache. Multi-query optimization enhances the performance of query execution in distributed database systems. Such a technique is used in [73]. The following steps are performed in this context:

1. Maintaining an intermediate data structure generated by queries.

2. Caching input data in memory.

3. Providing support for multi-threaded execution.

4. Use of hybrid shipping.

The proposed model reduces communication cost along with the response time. Multi-join query optimization is important for designing DBMS. The authors in [74] have proposed a

new algorithm based on parallel ant colony optimization for solving the problem of multi-join query optimization. The three factors for the implementation include (i) heuristic information, (ii) implementation of local and global pheromone update and (iii) design of state transition rules. The simulation results show that the parallel ant colony optimization behaves efficiently as compared to GA [74]. Numerous modern applications required complex event processing technology to analyze a multi-dimensional stream of big data that is available in real-time data feeds. A novel real-time event stream processing method known as Multi-Query Optimization Strategy (MQOS) is designed in [75]. The design is based on a triaxial hierarchy which consists of nested query patterns and operational hierarchies. The triaxial hierarchy also describes the relationship between the sub-expressions of queries. Based on the triaxial hierarchy, a cost-based heuristic is implemented so that an optimized QEP with minimum costs of operators and communications is found.

3.2 Fault Tree Models

There are several methods already present in the state of art in which fault diagnostic models are implemented. Some of these models are discussed in the next paragraphs.

Fault tree models are widely used in mechanical fault diagnosis. The reasons of fault occurrence are represented as an input event while the causes of faults are shown as the bottom events [76]. Qualitative trend analysis (QTA) is a data oriented technique to diagnose the process generated trends from noisy process data. These extracted trends are matched with the faulty trends already stored in the database. QTA is merged with Signed Directed Graphs (SDGs) in order to increase the benefits of timely diagnostic processing. The SDG–QTA based technique satisfies all the diagnostic and prognostic requirements in real time systems [77]. A dual-tree complex Wavelet Transform (WT) is employed to extract the features from the vibration signals. A Neural Network (NN) is used to classify the difference between healthy and faulty data in [36]. A discrete event model framework is implemented for active diagnosis in battery systems [78]. The normal messages and

fault messages are split into two different sets. An appropriate system algorithm is applied to find out which particular fault is causing the problem. Weighted fault diagnosis tree (WFDT) is modeled in [79] to implement the fault diagnosis mechanism. An extensive distributed information system is designed based on WFDT. Each node in a tree is an independent decision making unit. The proposed methodology is implemented in health care systems, and it provides accurate and timely fault diagnostic mechanism.

3.3 Graph Optimization

Graph partitioning is the process of dividing one graph into multiple subgraphs based on the similarities [80]. Subgraphs are generated by eliminating some of the nodes or edges from the parent graph. Graph partitioning strategies are categorized into two types: (i) local strategies and (ii) global strategies. The local strategy is based on the edge deletion, but it provides inefficient results as compared to the global strategies. On the other hand, global strategies are based on mathematical approaches which perform partitioning based on a network model [81]. Different heuristic approaches are adopted to achieve the best results in the shortest possible time [82]. A typical strategy based on heuristic rules is implemented in [83][84]. The algorithm partitions the graph into two equal portions and then iteratively improves the reduction by cutting edges. A multi-layer graph partitioning method based on heavy heuristics is introduced in [85] to optimize the graph.

Large graphs are partitioned by applying anti-roughening and optimization techniques. In [86], a graph optimization algorithm is proposed, which enables the optimization of large graphs and reduces memory consumption. The original graph is pruned and divided into multiple sub-graphs. Based on the root to leaf and leaf to root extraction optimization, the original graphs are optimized with small consumption of memory. In [87] authors study the problem of graph similarity search. They proposed a systematic method for the edit-distance based similarity search problem. The proposed method is applied by considering two lower bounds known as (i). partition-based bounds and (ii). branch-based bounds. A

uniform index structure known as u-tree is introduced for effective pruning and efficient query processing. Extensive experiments show that the proposed approach minimizes the overall query response time. Modern embedded systems are often complex due to their size and structure. This complexity makes the process of fault diagnosis and analysis difficult and time-consuming. In order to minimize this complexity, a directed fault propagation graph model is presented in [88]. The presented graph is optimized by considering the influence factor based on the quantitative assessment. After optimization, the redundant components are minimized, and the fault diagnosis process becomes simpler and less time-consuming.

3.4 Fault Detection and Diagnosis Using Graph Based Techniques

Dynamic Bayesian Networks (DBN) are graphical models which are employed for systems that have uncertainty and varying inference time. The nodes in these graphs represent the variables while the edges represent the relationship between these variables. DBN are used in different domains including risk management, cancer classification and reliability investigation [89]. However, there is not much research in the state of the art about the usage of DBN in the context of fault diagnosis for embedded systems [90]. Bond Graph (BG) is an effective method used for modelling complex systems. BG is able to present the components and structure of physical systems. This feature of BG enables the efficient analysis of fault values and online isolation of faults. A fault detection strategy is implemented for the autonomous vehicles known as RobuCar. The modelling of the vehicle is done by using the BG method. The BG method is also helpful for the systematic generation of Analytical Redundancy Relations (ARRs) [91].

A fault propagation method is implemented by using graph model. This fault oriented graph model is optimized by considering the fault influence factor and redundant components are deleted within the system. Due to this deletion the overall system is simplified and fault diagnosis and analysis becomes easy and scale able [88].

3.5 Real Time Databases Management Systems (RTDBMS)

Conventional relational database systems are unable to meet the stringent timing deadlines provided by safety critical systems. Therefore, RTDBMS are specifically designed for processing the data in a timely manner which is essential for safety critical systems [92].

RTDBMS are considered to be a salient branch of the conventional Database Management System (DBMS). Therefore the properties of traditional database systems are extended with real time processing. RTDBMS are employed in different domains including data acquisition systems and real-time control systems. RTDBMS have different design features due to the involvement of timing constraints. There is often a large amounts of data transference along with the requirements of data consistency and timing constraints. These RTDBMS use different metrics to evaluate the constraints based on the timing deadlines [93]. The quality of any RTDBMS can be measured by how often a transaction misses its deadline. Data consistency and temporal consistency both should be measured in case of missing deadlines. High throughput and low response times are further important objectives that RTDBMS should achieve [45]. During the last two decades, different technologies were developed for managing systems that deal with image processing data. Image processing should satisfy timing constraints in many applications such as autonomous driving or human/robot collaboration. The timing constraints in this context are often considered to be hard. Real- time control systems in which image processing is using RTDBMS play an important role as they are capable of dealing with the timing constraints. There are three types of timing constraints involved in these systems 1) image acquisition, 2) image transportation and 3) image processing time [94]. A real time DB was designed and tested on red black tree in [95]. Red black tree proposed by Rudolf Bayer is considered to be the most effective query tree providing the addition and deletion with defined deadlines. This feature makes the red black tree capable of maintaining the timeliness property of the real-time domain. Other common data structures are AVL trees that perform better

in terms of main memory usage.

Real-time database design is comprised of different algorithms that have to deal with memory efficiency and timely data manipulation. Some of the RTDBMS are not relying on the operating system for the memory, but they are capable of managing their memory requirements. Memory management methods of these RTDBMS includes: (i) heap array allocation, (ii) bitmap allocation and (iii) memory pool distribution. Concurrency control is also implemented by using lock based concurrency control methodologies [96]. Quality of Service (QoS) based timeliness and freshness techniques improve the quality of data and transactions in RTDBMS. The data freshness and timeliness of transactions can cause a conflict between different system requirements. The workload related to a particular transaction is not always precisely predictable. Heuristic algorithms are often used to implement the QoS based techniques. The co-scheduling problem of periodic applications is also studied in this context. Several methods are also proposed for the prediction of schedules in advance so that the deadline miss ratio is minimized [97]. RTDBMS need to process all the data in a stringent timing bound provided by the system. However, existing DB design and implementation mechanisms are susceptible overheads. To solve this problem a Single Input Single Output (SISO) feedback admission control scheme is introduced in [98]. The SISO controller in each node dynamically adjusts the amount of data that can be processed during a certain time period. Oracle Berkeley DB was used to implement this technique. Although the concept of RTDBMS is studied extensively in literature most RTDBMS considers only a centralized real time system. Due to this reason throughput, timeliness and the total amount of data processed within the deadline of the system is extensively decreased [99].

3.6 Worst Case Execution Time (WCET)

A measurement-based estimation method for the calculation of the WCET of programs in real-time systems is proposed in [100]. A Control Flow Graph (CFG) is designed for each

building block of a program. Execution profiles of each basic block along with their execution probabilities are extracted during the execution of a program. Afterward, a critical path is identified by calculating execution probabilities of all feasible paths in the CFG.

As the number of applications in embedded systems is increasing in all aspects of life, the performance issues in real-time systems gain a lot of attention from researchers and practitioners. In the case of real-time systems, the calculation of accurate WCET is an important problem. The WCET analysis method described in [101] is known as CRYINGCAT. CRYINGCAT is independent of the compiler model. It calculates the WCET by collecting information from the CFG, cache access, and pipeline access. Hardware modeling and accurate path selection are two major problems that hinder the calculation of a precise WCET. Authors in [102] have successfully combined the statistical analytics with run-time measurements of a program. The proposed technique is tested and implemented on real hardware. Conventional methods depend on the technique of static timing analysis for computing the Worst-Case Response Time (WCRT) of tasks in real-time systems. Multi-core real-time systems face concurrent task executions, semaphore accesses, and task migrations. It is difficult to obtain the upper bound of execution time in multi-core real-time systems. A probabilistic estimation method is presented in [103]. This method solves the problem of upper bound calculation for WCRT. WCRT is estimated for a set of tasks with different times. The proposed technique classifies the data with sample size equalization.

Traditional WCET analysis techniques are not appropriate for control systems comprised of modern multi-core processors. In [104], authors have proposed modifications for already existing tools that calculate the WCET for multi-core processor based control systems. For compositional architectures, the proposed modifications introduce time-effective analysis along with predictability. Authors in [105] have proposed a technique for the applications running on multiprocessor systems-on-chip. They have considered average case and worst case delays. It is evident from the results that the proposed technique reduces the average-case delay dramatically while keeping the worst case delay as small as possible.

The proposed method combines average and worst-case optimization techniques for real-time systems. The proposed method also provides a guarantee for worst-case predictability.

Authors in [106] have proposed a method for low-level timing analysis of the program. This method is based on measurements of execution times of programs. These programs are executed on the real architecture. The essence of this method is to derive a system of linear equations from a limited number of timing measurements for an instrumented version of the program. The main advantage of this approach is that it does not require architectural modelling. Therefore, the risk of a discrepancy between the system model and the real system is avoidable. As compared to the non-exhaustive measurement-based approaches this method is considered to be more structured and gives complete coverage in terms of the program paths. It is difficult to calculate the tight upper bound of the worst-case response time in distributed real-time embedded systems. There are numerous factors which may cause this complexity. These factors include (i) variations of the execution time of tasks, (ii) jitter of input arrivals and (iii) scheduling anomalies. In [107] authors, propose a novel solution based on ILP (Integer Linear Programming). In the proposed technique, a set of ILP formulas are formulated, while supporting model flexibility. This problem is solved holistically to achieve tight upper bounds. To mitigate the time complexity of the ILP method, static analysis based on a scheduling heuristic is added in the proposed technique. Experiments show promising results that give tight bounds in an optimized time.

Due to the increase in the complexity of embedded systems, it is difficult to verify that all the system requirements are fulfilled under all possible execution scenarios. There are numerous assumptions (e.g., WCET, maximum jitter, minimum activation period) that are considered in this context. It is not always possible that these assumptions are always fulfilled at run time. Because of these reasons, run-time monitoring and run-time verification become an effective alternative to the conventional offline verification. In [108], authors present four different implementations of a run-time monitoring framework for safety-critical systems. Two implementations of this framework are written in Ada and

are appropriate for the development of high integrity systems. The third implementation is written in C++ and is compatible with the majority of operating systems. The fourth and last implementation is a kernel module for Linux and saves the system from memory faults.

Researchers have proposed various methods for the calculation of WCET in safety-critical systems [109]. Typical Worst-Case Analysis (TWCA) is introduced to minimize the frequency of undesired behaviors based on the concept of weakly hard constraints [110]. The ILP based method is introduced in [111]. This method is suitable for the systems where each task has a different deadline for its completion. The proposed approach addresses the static-priority non-preemptive (SPNP) and static-priority preemptive (SPP) scheduling policies. A few control applications endure the violation of deadlines to some extent. The calculation of overly pessimistic WCET is not feasible for these applications. Some systems are greatly affected by the average response time as compared to the WCET [112]. Probabilistic WCET analysis techniques use a probability distribution of execution times with a certain level of confidence. A method for computing, probabilistic bounds on the execution times of a task is proposed in [113].

Research work in embedded database systems is primarily focused on the algorithms related to the concurrency control and scheduling of transactions [114]. The evaluation of these algorithms is based on the number of deadlines missed by a particular transaction. The efficiency of these algorithms is affected by the number of transactions processed concurrently. There are numerous algorithms that give solutions for the processing of concurrent transactions [115]. It is crucial that all tasks should fulfill their deadlines provided by the system. It is difficult to calculate the accurate WCET for the database transactions in event-driven hard real-time systems [116]. Authors in [117] proposes an Adaptive Total Bandwidth Server (ATBS) algorithm for minimizing the response time of aperiodic tasks with different execution times. This method reduces the response times of aperiodic tasks by using predictive execution times instead of WCET. The proposed ATBS and ATBSRR techniques are compared to the total bandwidth server technique. Results show that the

average response times for aperiodic tasks are reduced up to 48% [117]. Authors in [118] have proposed a novel analytical method known as scheduling time bound analysis. The proposed method is used for finding a tight upper bound of the worst-case response time in a distributed real-time embedded system. Tasks having different execution times and jitter of input values are also studied in the context of the proposed technique. By analyzing the graph topology and worst-case scheduling scenarios, the conservative scheduling time-bound of each task is measured. The proposed method supports an arbitrary mixture of preemptive and non-preemptive processing elements.

CHAPTER 4

SYSTEM MODEL

This chapter elaborates the system model of this research. The system model has two components, an (i) architecture model and an (ii) application model. Architecture model elaborates the physical representation of our system. The application model represents our DMG which is used for the fault detection and diagnosis.

4.1 Architecture Model

Our architectural model is based on the physical representation of our proposed system. Our architecture model consists of sub nodes comprising of R routers and P processors with L links between them. The processors are connected with each other by using bidirectional communication links. The architecture of this system S is represented by Eq. 4.1:

$$S(P,R,L) \tag{4.1}$$

Where $p_i \in P$ is a homogeneous processor and each $r_i \in R$ represents a router in S. The bi-directional link $l_{p_i r_i} \in L$ shows a connection between processor p_i and router r_i whereas each $l_{r_i r_j} \in L$ represents a link between routers r_i and r_j. The designed system architecture has the following characteristics:

1. The system has homogeneous processors with heterogeneous links (having different bandwidths).

2. Each $p_i \in P$ has its own sensor attached to it. This sensor is providing the input data to the particular process.

3. The system has its own real time database. Each $p_i \in P$ stores the input data from the sensors in a RTDBMS known as PSQL.

4. The database stored at one p_i is replicated on another p_j if the same data is required.

5. The processors are designated for the processing of queries only and will not take part in any kind of communication. Similarly, routers are only available for the transmission of messages.

6. Communication among different p_i is carried out via a deterministic protocol such as Time Division Multiple Access (TDMA) and only one message is transmitted through a link during a given period of time.

7. If two FDQs are assigned to the same end systems then their communication cost is negligible. This constraint is taken into account because in many distributed systems, remote communication is more costly as compared to local communication.

8. If two FDQs that are dependent on each other and are assigned to two different processors, then they have a communication cost C_C associated with them. Therefore this C_C is calculated by considering the path (followed from one p_i to another p_j) and the size of the message that is being transferred across that path.

9. Once a FDQ is assigned to a processor, it executes all its periodic iterations on the same processor.

10. Similarly, a communicating FDQ executes all its iterations on the path that has been assigned to it.

    There is no restriction about the network of topology in our proposed system architecture and it can use any topology along with an arbitrary number of processors and communication links between these processors. Our architecture model is depicted in Fig. 4.1.

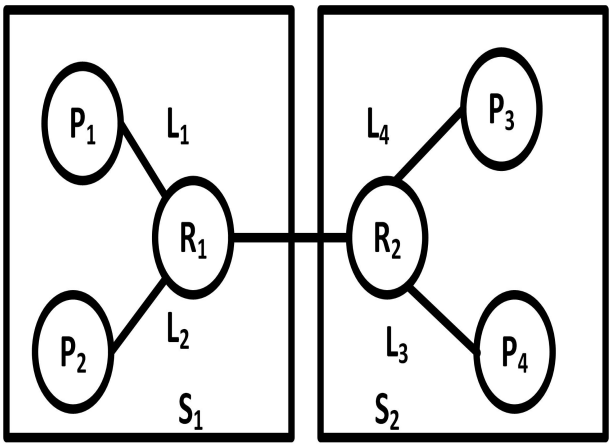


Figure 4.1: Example of Architecture Model

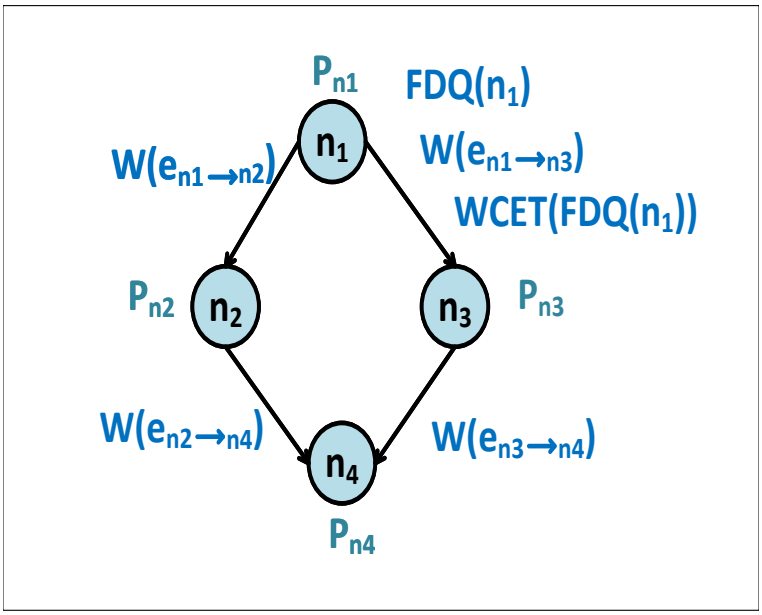


Figure 4.2: Application Model: Structure of Proposed DMG

4.2 Application Model

Active diagnosis is different from passive diagnosis. It involves the retrieval of diagnostic information and analyzing it in a timely manner. In order to enhance the system reliability, it is important to retrieve the right diagnostic symptoms and analyze them so that recovery actions can be implemented without missing the deadline given by the systems. Therefore, it is vital to build an authentic and precise diagnostic model which can represent the FDQs and enable the system to extract and process the information based on the fault that occurs in the system.

Therefore after analysing the different state of the art methods, it is clear that there are numerous methodologies which are used for active diagnosis in real-time systems. Hence for enhancing the reliability of real-time systems (by computing diagnostic information at run-time for fault recovery), we need a fault diagnosis mechanisms in the system. Considering this aspect of active diagnosis, we have designed a multi-query graph for our system. This graph is formally termed as Diagnostic Multi-Query Graph (DMG) and has been previously used by the authors in [119]. A DMG is similar to a directed task graph with the exception that it is implemented using a real-time database. In our DMG, each root node is associated with a set of sensors that provide continuous input data according to their periods. An example of proposed DMG is shown in Fig. 4.2.

This input data is stored in a real-time database. The sensors utilize local-error detection methods to formulate diagnostic facts that are further used to identify symptoms, faults and to propose respective recovery actions. These diagnostic facts are realized as queries in this real-time database and are executed repeatedly to keep the diagnosis up-to-date with the input data. In the DMG, the FDQs are represented by nodes and the relationship between the FDQs within the real-time database is represented by edges. The nodes are labeled with the $WCET$ of their associated FDQs. Since we are dealing with real-time systems, each node is cyclic with a strict period [120]. A strict period means that if a task A has a period

P_A then the difference between the starting points of two of its consecutive iterations should be equal to P_A. Here, nodes are termed as diagnostic features (without incoming edges), symptoms (with both incoming and outgoing edges) and faults (without outgoing edges) respectively. The edges are labeled with the amount of data d the nodes are transmitting to the database. As a FDQ repeats its execution each time with a different set of input data, it is possible that its corresponding child FDQ requires data from one of its previous or following repetitions rather than its current one.

4.2.1 Diagnostic Mutli-Query Graph (DMG)

The diagnostic system model requires the optimization of diagnostic queries in a manner that they can be completed in the pre-defined deadline of the system. These FDQs are scheduled on the heterogeneous processors for calculation of their makespan. The application model is represented by a diagnostic graph comprised of FDQs. This graph is formally termed as a diagnostic multi-query graph (DMG). In our DMG, each FDQ is unique and is linked with a sensor or a specific set of sensors. The sensors provide data in a periodic manner, which is then stored in the real-time database and is timely replicated onto the processors to ensure the distributed execution of the FDQs. The FDQs are used the identification of faults. These FDQs are based on the features and symptoms extracted directly from the sensors data. Our diagnostic queries are represented in SQL format. Each FDQ is executed periodically to keep the database up-to-date with the sensors data. In a DMG, each node represents a single FDQ and each edge specifies the relationship between two FDQs through the real-time database i.e. the precedence constraints and the parent-child relationship between the FDQs. These FDQs are represented in the SQL format comprising SQL operations such as select, insert and join.

The nodes are termed as features (nodes without incoming edges), symptoms (nodes with both incoming and outgoing edges) and faults (nodes without outgoing edges) respectively. Since the sensors are sending data periodically, it is essential for the FDQs to be

executed after continuous time intervals to keep the analysis up-to-date. Therefore, each FDQ has a strict time period. Since the cyclic execution of FDQs results in various iterations, it is possible that the child node requires output data from the previous iteration of a parent node rather than the present one.

The overall DMG is defined as follows

$$DMG = (N, E) \tag{4.2}$$

$$N = (n_1, n_2, ..., n_i) \tag{4.3}$$

$$E = (e_1, e_2, ..., e_i) \tag{4.4}$$

$$e_i = (n_p \rightarrow n_c) \tag{4.5}$$

Where a parent node n_p has a precedence constraint over a child node n_c. Each vertex $n_i \in N$ is a non-divisible periodic task. Each directed edge $e_{n_i n_j} \in E$ illustrates the precedence constraint between $n_i \in N$ and $n_j \in N$ such that n_i is the parent vertex to n_j. A positive weight $WCET(FDQ(n_i)$ represents the WCET of vertex $n_i \in N$, $FDQ(n_i)$ describes the FDQ associated with this vertex while P_{n_i} represents its time period. Since we are dealing with periodic tasks, for a DMG represented by Eq. 4.2, each vertex $n \in N$ will repeat its execution after an interval P_n. This interval is termed as the time period and it is the exact time elapsed between two consecutive iterations of n [121]. $W(e_{n_i \rightarrow n_j})$ represents the amount of data that is transferred from the parent node n_i to the child node n_j along the edge e. This amount of data effects the communication cost between the tasks as larger data will take more time for the transfer. The subsequent child node $n_i \in N$ cannot start its execution before all of the data between these particular executions of n_i has been successfully transmitted to its assigned processor. The size and structure of the DMG depends on the application. A DMG can comprise any number of nodes and edges.

4.2.2 Scheduler

The proposed diagnostic framework depends on a scheduler, which is responsible for the temporal and spatial allocation of resources for the time-triggered execution of FDQs and for the communication between the processors. The scheduler used in our work was introduced in [122]. It is calculating the makespan of our DMG. The DMG along with the (i) $WCET(FDQ(n_i)$ and (ii) the values $W(e_{n_i \rightarrow n_j})$ is provided as an input to the scheduler. The scheduler calculates the makespan of the DMG by assigning each node of the DMG to processors depending on the availability of processing resources. The scheduler also assigns communication resources by determining the paths on the network. A detailed description of the scheduler is out of the scope of this thesis.

CHAPTER 5

CLASS-BASED QUERY-OPTIMIZATION FOR MINIMIZING THE WCET OF THE DMG

This chapter describes the class based query optimization technique that is used for the minimization of the WCET of our DMG comprised of FDQs. The proposed algorithm is comprised of several steps which are elaborated in this context.

5.1 Basic Algorithm

This section elaborates the algorithm designed for the implementation of our technique. Class of the FDQ is specified on the basis of its type. We have considered two types of FDQs (i). select and (ii). join. Each FDQ selects the different access method depending on its type. If the FDQ is join then the best join order is also selected for the purpose of query optimization. For each FDQ the best QEP is selected as a part of optimization technique. Algorithm 1 represents the pseudo code representation of our alogrithm.

Algorithm 1 Class Based Query Optimization: CBQO

Input:DMG comprised of Fault Queries
Output:Optimized DMG with reduced WCET

1: Define classes for queries
2: Calculate WCET for each Query Q
3: **while** (DMG(N)!=0) **do**
4: Extract SQL queries from DMG
5: Select class for each query
6: Select the access method for query
7: Calculate selectivity factor Sf for join queries
8: Select join order with smallest Sf
9: **end while**
10: return DMG

5.2 Brief Details of Algorithm

1. Class of the FDQ is specified on the basis of its type. We have considered two types of FDQs (i) select and (ii) join.

2. Each FDQ selects the different access method depending on its type.

3. If the FDQ is join then the best join order is also selected for the purpose of query optimization.

4. For each FDQ the best QEP is selected as a part of optimization technique.

5.3 Detailed Explanation of the Algorithm

The proposed DMG used for the implementation of our technique is shown in Fig. 4.2. Each node of the DMG has a $FDQ(n_i)$ associated with it. $FDQ(n_i)$ is written in SQL format and comprised of different SQL operations including (i) selection (σ), (ii) projection (Π) and (iii) join ($\bowtie$). These FDQs are executed over the database created in Pervasive SQL (PSQL). For minimizing the overall WCET of FDQ, it is important to optimize them. The optimization enables the FDQs to be completed in the stringent timing deadline provided by the system. Therefore, it is important to notice that decreasing the WCET of the $FDQ(n_i)$ is helpful for meeting the deadline. For each $FDQ(n_i)$ different QEPs are generated and the QEP with the minimum cost in terms of time is selected. The proposed algorithm has three steps:

1. FDQs are divided into different classes. Each class is based on the type of FDQ. The FDQ can be a select or join query. For each $FDQ(n_i)$ the appropriate class is selected.

2. On the basis of the type of class that is selected for $FDQ(n_i)$, the best access method with the minimum execution time is selected.

Table 5.1: Classes for Simple Query Types

Name	Query Type	Operation	Access Method
A	Simple queries	select	Full table Scan
B	Search Queries	$<, >, =$	B-tree scan

Table 5.2: Classes for Join Query Types

Name	Query Type	Join Access Method
C	Inner table of join fits in the memory	Nested loop
D	Join is an equi-join	Hash
E	Not an equi-join	Sort Merge

3. For FDQs belonging to the join class, a join optimization technique is applied. The best join order for each $FDQ(n_i)$ is selected by considering the selectivity factor (S_f). The S_f for a particular join FDQ is the time taken by $FDQ(n_i)$ to find the number of tuples present within that join order. The join order with the smallest S_f (i.e., join combination which takes minmal time to read the require tuples) is considered.

There are multiple access methods which are used by modern optimizers during the process of query optimization. These access methods include B+ tree scan and full table scan. Our methodology selects the access method for the $FDQ(n_i)$ on the basis of query class in which it falls. Classes defined for the categorization and access methods of simple select queries are shown in Table 5.1. The class categorization along with access methods for join queries are shown in Table 5.2.

Before selecting the join order for the FDQ, it is important to find the access method for the relations in the query. The access method for the join FDQ is selected by using class based categorization from Table 5.2. After the selection of the access method the best join order is selected. the selection of the best join order is an important research issue [123]. For this purpose the Left Deep Tree (LDT) is generated on the join predicate of the SQL query. LDTs are considered to be the best tree because they minimize the overall

memory utilization of the query. Also in case of real time systems available memory is often restricted, so this is an important constraint to be considered.

The best join order of the relations present within the join based $FDQ(n_i)$ is determined by calculating the S_f. S_f is the ratio between the total number of tuples in a relation and the tuples required by $FDQ(n_i)$ during its execution. Therefore S_f is calculated for both read and write operations performed by $FDQ(n_i)$. The S_f is based on the parameters in Table 5.3 extracted from database.

Table 5.3: Parameters extracted from Database

Parameter	Description
R	Relation in a database
t_i	Set of all tuples over relation R
$(R_1 \bowtie R_2 \bowtie ... \bowtie R_i)$	join order relations within query
$Q(t_i)$	Query predicate/condition over the tuple t_i
$D_r((R_1 \bowtie R_2 \bowtie ... \bowtie R_i)t_i)$	Disk reads required for join order
$D_w((R_1 \bowtie R_2 \bowtie ...R_i)t_i)$	Disk writes required for the join order

$$A = [n(D_r(R_1)) + n(D_r(R_2)) + ... + n(D_r(R_i))] \tag{5.1}$$

$$B = [n(D_w(R_1)) + n(D_w(R_2)) + ... + n(D_w(R_i))] \tag{5.2}$$

$$C = n(D_r(\exists t_i \in (R_1 \bowtie R_2 \bowtie ...{}_i)Q(t_i))) \tag{5.3}$$

$$D = n(D_w(\exists t_i \in (R_1 \bowtie R_2 \bowtie ... \bowtie R_i)Q(t_i))) \tag{5.4}$$

Eq 5.1 and Eq 5.2 represent the number of disk reads (D_r) and disk writes (D_w) required by the full relation scan present in the Q_{ni}. Eq 5.3 and Eq 5.4 represents that there exists a tuple t_i in the relation order $(R_1 \bowtie R_2 \bowtie ...R_i)$ on which the predicate $Q(t_i)$ is true. Variables C and D give the number of D_r and D_w required for the selection of tuples based on the predicate $Q(t_i)$ within a selected join order. The S_f for each join order is defined as follows:

$$Sf_j = \frac{A+B}{C+D} \tag{5.5}$$

The solutions space Sp is based on the solution set of different Sf for different combinations of joins in $FDQ(n_i)$.

$$Sp(Sf) = (Sf_1, Sf_2, Sf_3 ... Sf_i)$$

The best Sf for the join order is as follows:

$$Sp(Sf) = min(Sf_1, Sf_2, Sf_3 ... Sf_i) \tag{5.6}$$

So overall the optimized fault diagnostic $FDQ(n_i)$ has the best access method and the best join order with minimum Sf.

5.3.1 Estimation of Worst Case Execution Time (WCET)

An estimation of the WCET is denoted as $EWCET(FDQ(n_i))$ is provided as an essential input for query optimization. There is a difference between the WCET and the estimation of the WCET. The adopted term EWCET is the derivation of estimates for the maximum execution time of an FDQ. A measurement-based method is used for calculating the EWCET of a simple query [124]. By measuring the execution time for queries during execution with different input data, the maximum and minimum limits based on the observed execution time are derived. Measurement of these fault queries generate the estimated WCET but not the actual WCET. $EWCET(FDQ(n_i))$ is the CPU time dissipated by this $FDQ(n_i)$ until its execution is completed.

5.4 Illustrative Example

This section elaborates an example scenario in which different FDQs are designed and the proposed optimization is applied. These FDQs are executed on a database created in Pervasive.SQL (PSQL). The query optimization module is implemented by using the JDBC interfacing of PSQL. The database contains relations which store data about a car's sensors (i.e., data-set taken from vehicles). PSQL is used because it has numerous benefits in the real-time scenario as compared to the conventional database management systems. The proposed query categorization technique allows to find the query class and access method for the FDQ. This technique minimizes the overall EWCET by up to 56%. The features included in $FDQ(n_i)$ are generated from the vehicles data. An example FDQ is represented by Q_1 which has the following features and symptoms.

Q_1:Select Fault from OxygenSensors O JOIN EngineSensors AS E on O.TempID=E.TempID JOIN TemperatureSensors AS T ON T.TempID=E.TempID where $O.AirRatio > 0.1$ and $E.FuelRatio > 0.2$ and $O.OilTemp > 0.3$.

5.4.1 Features

The three features of the vehicles are added in Q_1. These features are (i) oil, (ii) temperature, (iii) fuel ratio and (iv) air ratio.

5.4.2 Symptoms

Symptoms are values which are generated by the sensors in case of fault occurrence. These values are read and written in the database every 15ms. Symptoms in the presented Q_1 are the values present in the where clause, which are AirRatio and FuelRatio and OilTemp. Each symptom value present in Q_1 is compared with the values stored in the database. If the value of symptoms is greater than a certain threshold provided by the sensors, then it is assumed that there is some fault in the system at the current point of time. Based on both

features and symptoms, Q_1 is executed, and the fault is diagnosed. The proposed algorithm implements the optimization methodology in the following sequence.

- The class of Q_1 on the basis of its type is selected. The example Q_1 belongs to the class D where the joins are equi-joins, so the selected access method will be the hash join method.
- The maximum observed execution time is the time required by the Q_1 to execute with the selected access method and the best join order.

Three tables oxygensensors(O), EngineSensors(E) and TemperatureSensors(T) are join in the Q_1. All the join orders are considered and the best one is selected on the basis of the value of the Sf. Different join orders are generated and Sp is created . Sp is searched exhaustively by the query optimization module for finding the minimum Sf ratio. As Q_1 has three joins, there are six possible LDTs. The cost for each join order is calculated on the basis of the Sf.

Table 5.4: Cost of Join Orders in Q_1

Access method	Join Orders	Sf	EWCET (ms)
	$T \bowtie (O \bowtie E)$	27%	567
	$O \bowtie (T \bowtie E)$	31%	577
	$T \bowtie (E \bowtie O)$	43%	643
	$E \bowtie (O \bowtie T)$	56%	721
Hash Join	$O \bowtie (E \bowtie T)$	87%	910
	$E \bowtie (T \bowtie O)$	51%	702

Table 5.4 represents the EWCET and Sf for Q_1. On this basis the join order with minimum Sf is selected. The minimum EWCET is considered to be the best selection for example Q_1.

5.5 Results

This section represents the results of the optimization technique performed and tested over different types of FDQs (synthetically defined) including select and join. The example data is created synthetically and use to populate the database. The data tables with different sensor data are populated by using insert queries. Almost 80,000 sensor values are inserted into database to test the validity of our approach. These sensor values are collected from different websites [125].

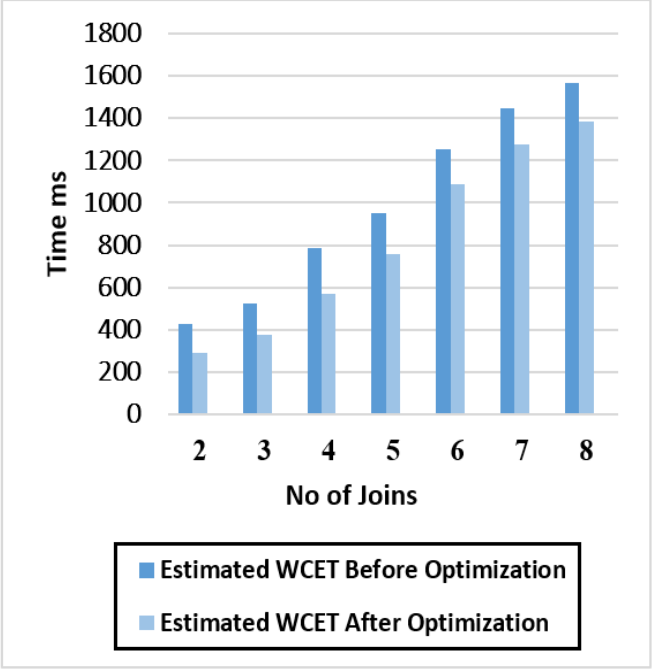


Figure 5.1: Comparison of EWCET of FDQs Before and After Optimization

Fig. 5.1 presents the comparison between the EWCET of the FDQs before and after optimization. The different numbers of joins are taken into consideration for analyzing the performance aspect of proposed method. It is evident from results that the EWCET for fault diagnostic queries is decreased after optimization is applied. In order to ensure the fault diagnosis to be completed within the time bound of the system, the EWCET of FDQs is measured before and after the optimization is applied. It is evident from the results

that after the optimization, all the FDQ are enable to minimize their EWCET, which is the most important objective to achieve for our real-time system. So overall optimization of these FDQs helps the system to meet their deadline. Decreasing overall EWCET is vital in our proposed real time system, so that the fault of a system can be diagnosed before any catastrophe may occur.

CHAPTER 6

MINIMIZING THE MAKESPAN OF DMGS USING GRAPH PRUNING AND QUERY MERGING

This chapter describes our proposed technique for minimizing the makespan of DMGs. This technique focuses on the optimization of DMGs and FDQs both.

Algorithm 2 Graph Pruning (GP)

```
1:  procedure GRAPHPRUNING(DMG)
2:      for each N_i ∈ DMG
3:      while Depth(DMG) ≠ max do
4:          Access the DMG with BFS
5:          Calculate(no(nodes) ∈ level(DMG))
6:          level ← maxnoofnodes
7:          while level ≠ 0 do
8:              for each FDQ(N_i) ∈ level
9:              if p(FDQ(N_i)) = pneighbour(FDQ(N_i)) then
10:                 QueryQueue ← Query(N_i)
11:             else
12:                 break
13:             end if
14:         end while
15:         while QueryQueue ≠ 0 do
16:             for each FDQ(N_i) ∈ QueryQueue
17:             if Op(FDQ(N_i)) = Op(FDQ(N_{i+1}) then
18:                 if DT(FDQ(N_i)) = DT(FDQ(N_{i+1}) then
19:                     MergeQuery(FDQ(N_i), FDQ(N_{i+1})
20:                     Delete(QueryQueue(FDQ(N_{i+1})))
21:                 end if
22:                 if DT(FDQ(N_i)) ≠ DT(FDQ(N_{i+1}) then
23:                     MergeQuery(FDQ(N_i), FDQ(N_{i+1}, Join)
24:                     Delete(QueryQueue(FDQ(N_{i+1})))
25:                 end if
26:             end if
27:             NewDMG ← QueryQueue(FDQ(N_i))
28:         end while
29:         GRAPHPRUNING(NewDMG)
30:     end while
31: end procedure
```

6.1 Overview of Graph-Pruning and Query Merging

Algorithm 2 applies DMG based pruning for the nodes n_i. Sets of constraints are defined for applying pruning. $FDQ(n_i)$ which is a FDQ is attached to each node. The query predicates $FDQ(n_i)$ in pruned nodes are not deleted but they are joined with their closest neighbour n_j which can be at the same level or depth. There is a data transfer along each e_i. If n_i is deleted then its related e_i is also deleted. But the data along this e_i is also transferred to the same n_j which has the $Q(n_i)$ from the deleted n_j. In this manner the data transfer cost is also minimized because now there is a reduction of e_i from parent to child node. After the pruning of $DMG(N)$ is completed the next step is the optimization of the final $FDQ(n_i)$ attached to each n_i. For selecting the most optimized version of $FDQ(n_i)$ the best access method and the best join order are determined. Therefore, the best QEP with minimum execution time is generated for each $FDQ(n_i)$.

In order to minimize the processing of data for all n_i except the root node, after the completion of query execution, a new data table is formed. These new data tables store the values which are transferred to child n_i after the execution of $FDQ(n_i)$. In this way, the data tuples which are only required by the child nodes are transferred from their parent nodes. Pervasive SQL (PSQL) is selected as a DBMS in order to get advantages of a real time data management system. Relations present in the database are populated by executing insert queries. The data-set is based on the data generated from vehicles sensors. The optimized DMG along with the WCET of each FDQ is provided as an input to a scheduler in order to calculate the makespan of the DMG.

6.2 Detailed Explanation of Algorithm

The proposed algorithm focuses on two aspects of optimization (i) graph based pruning and (ii) query based optimization (selection of best QEP). In this section graph pruning based optimization is described first and then the description of query based optimization

is provided. The steps in the proposed method are shown in Fig. 6.1

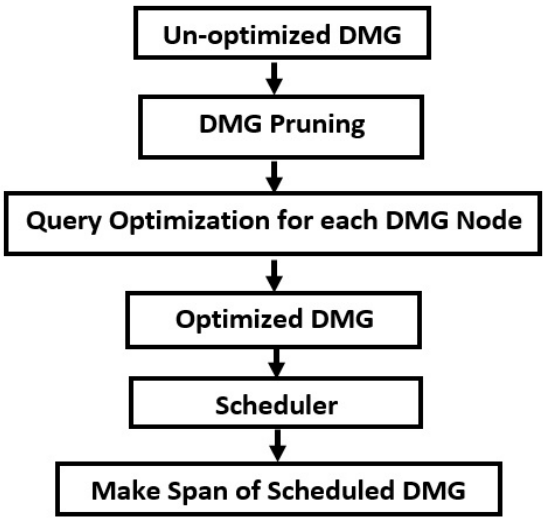


Figure 6.1: Steps in Proposed System

6.3 Graph Optimization

The DMG used in our algorithm is shown in Fig 6.2. The primary objective of graph pruning is to optimize the DMG by deleting different nodes based on constraints and minimizing its size. These constraints are helpful for the pruning of the DMG and minimize the overall makespan. The deletion of n_i is not straightforward as each n_i has a $FDQ(n_i)$ attached to it. Whenever any n_i within the DMG is deleted, its $FDQ(n_i)$ should be transferred and merged with its closest neighbour n_j. If the constraints that are defined for the merging of $FDQ(n_i)$ and the deletion of n_i are not met, then it is not possible to complete the delete process for the DMG. The defined constraints for DMG pruning and query merging from Table 6.1 are elaborated in following.

- **Constraint** C_1: It is the most important constraint that should be satisfied as a part of a DMG optimization algorithm. In order to delete n_i and merge its $FDQ(n_i)$ with its closet node n_j, it is vital that both n_i and n_j should be present at the same level of

Table 6.1: Graph Pruning Constraints

Constraints	Description
C_1	$Level(n_i) = level(n_j)$
C_2	$P(n_i) = P(n_j)$
C_3	$NotDel(\forall C(n_i) \in Level(DMG))$
C_4	$SO(FDQ(n_i)) = SO(FDQ(n_j))$
C_5	$DT(FDQ(n_i)) = DT(FDQ(n_j))$

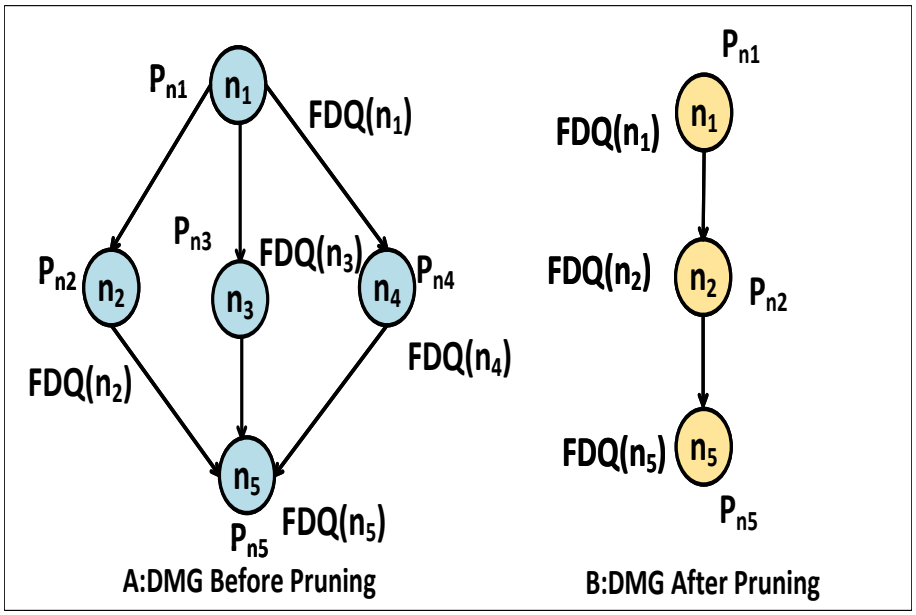


Figure 6.2: Representation of DMG Before and After Pruning

DMG. The Depth level of the the DMG is considered to be the distance from the root node to the n_i and n_j. If the nodes n_3 and n_4 are deleted in Fig. 6.2A then $FDQ(n_3)$ and $FDQ(n_4)$ can only be merged with $FDQ(n_2)$ because all of the three nodes n_2, n_3 and n_4 are at the same depth level in the DMG. Therefore according to the constraints it is not feasible that $FDQ(n_4)$ is deleted and is merged with the node $FDQ(n_5)$.

- **Constraint** C_2: The node n_i which is deleted and n_j which is accepting the $FDQ(n_i)$ must have the same parent. For example, n_2, n_3 and n_4 all have the same parent n_1 in Fig. 6.2A.

- **Constraint** C_3: All children nodes belonging to one parent can be deleted. If n_3 and n_4 are deleted then n_2 cannot be deleted. Otherwise the FDQs in these nodes are lost and the fault cannot be diagnosed.

- **Constraint** C_4: Constraint C_4 and C_5 are based on the FDQs attached with each DMG(N). Each query $FDQ(n_i)$ is split into parts. The query splitting is based on the two query clauses: (i) SQL operation (SO) and (ii) database table name. After the $FDQ(n_i)$ is split into parts, each query clause is matched with the $FDQ(n_j)$ of its closest node with which it has to be merged with. $SO(FDQ(n_i))$ represents the SQL operation present in each DMG(N). For merging the queries the "select" operation is considered. If the two queries $FDQ(n_i)$ and $FDQ(n_j)$ have two different SO namely "select" and "insert" then they cannot be merged. If the $SO(FDQ(n_3)) = SO(FDQ(n_4))$ then queries of n_3 and n_4 can be merged (see Fig. 6.2A).

- **Constraint** C_5: After the constraint C_4 is satisfied $DT(FDQ(n_i))$ represents the database tables present in each $FDQ(n_i)$. Two cases are considered on the basis of database tables for merging the queries.

6.3.1 Case 1

In the first case the assumption that is considered is $DT(Q(n_1)) = DT(Q(n_2))$. It is seen that both FDQs have the same database tables for their execution. If $DT(Q(n_2)) = DT(Q(n_3)) = DT(Q(n_4))$ then the process of deleting n_3 and n_4 and merging $Q(n_3)$ and $Q(n_4)$ with $Q(n_2)$ is allowed. Fig. 6.3A represents the example for Case 1.

6.3.2 Case 2

In this case it is assumed that $DT(Q(n_1)) \neq DT(Q(n_2))$. $Q(n_1)$ and $Q(n_2)$ do not execute on the same database tables. These FDQs are merged by employing the join operation. Fig. 6.3B shows an example for Case 2. The propose graph optimization algorithm perform its

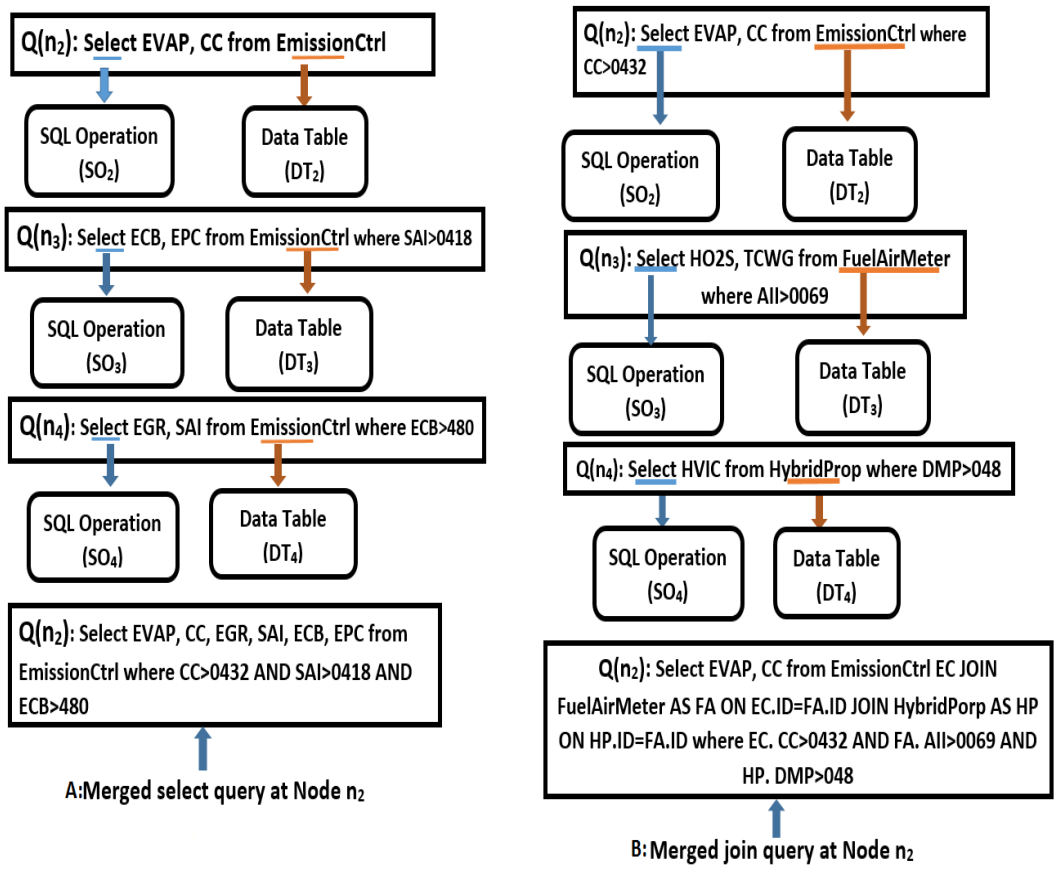


Figure 6.3: Query Merging: Case 1(A) and Case 2 (B)

operations in a cyclic manner. If the DMG has any set of constraints that it can satisfy then graph pruning is iteratively performed. The DMG generated after applying the pruning algorithm is shown in Fig. 6.3B.

6.4 Query Optimization

After the graph pruning based optimization is finished, the next step is to employ the optimization for all $FDQ(n_i)$ in DMG(N).

1. The best access method is found for all $FDQ(n_i)$. Access methods can be Table Scan (TS) or Index Scan (TS). The access method which returns the resultant tuples with the minimum execution time is considered as the best solution.

2. After the selection of the best access method the best join order for all join queries is also found. Join methods can be off various types including Equi-join, Merge Join (MJ) and Hash Join (HJ). The method with minimum processing time is always taken into account in this scenario.

3. For $FDQ(n_i)$ which have more than two join operations, the best join order with minimum Sf is considered. Different join orders for one $FDQ(n_i)$ are determined by using the LDT [14]. The LDT is considered to minimizes the number of join orders present in the Sp. After that for each join order the Sf is calculated [15]. The Sf can be calculated as follows:

$$S = \frac{|R_1 \bowtie R_2|}{|R_1| \cdot |R_2|} \tag{6.1}$$

4. After the selection of the best access method and the best join order, different QEP are created for each FDQ. The best QEP for the FDQ $Q(n_2)$ in Fig.6.3B is shown in Fig. 6.4B. The cost of each QEP is calculated by estimating the total execution time it takes for its completion.

6.5 Calculation of WCET

The worst case execution time $WCET(Q(n_i))$, is an important input for real time systems [122]. The WCET in this context is an input for calculating the makespan of DMG. The WCET is the maximum time $FDQ(n_i)$ may take during its worst case. For calculating the WCET of each query, different QEPs are considered. The QEP with the maximum time to process the tuples is selected to be the worst one. This WCET for each $FDQ(n_i)$ is given to our scheduler for calculating the maekspan. The worst case QEP for $Q(n_2)$ in Fig. 6.3B is shown in Fig. 6.4A.

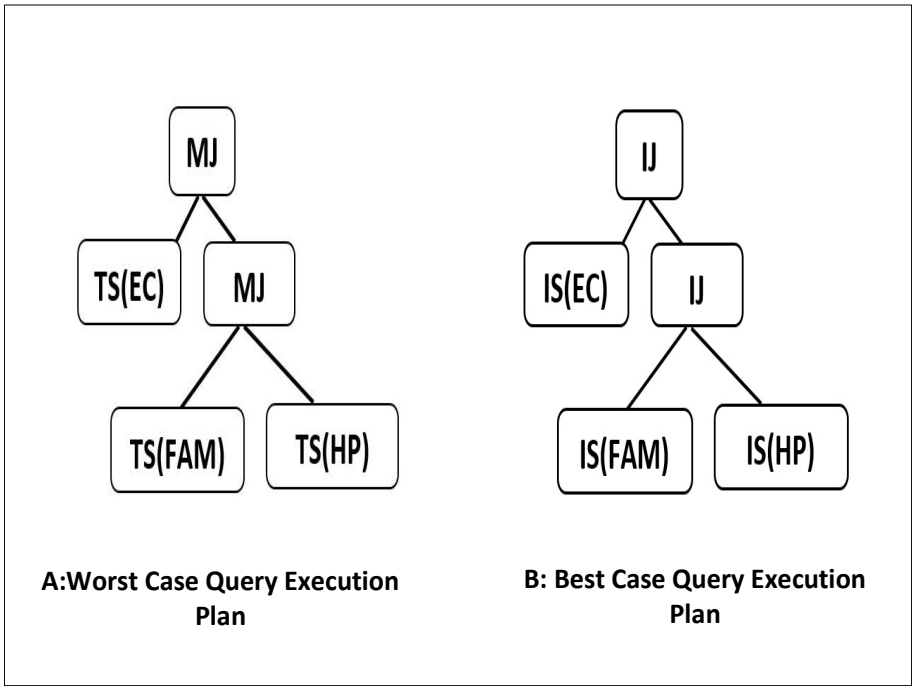


Figure 6.4: Worst Case and Best Case QEP for FDQ $Q(n_2)$ from Fig. 6.3B

6.6 Illustrative Example

This section elaborates an illustrative example on which the proposed algorithm is applied and tested.

6.6.1 Queries in DMG Before Optimization

The data from sensors is received by the root Node A. The FDQs that store the sensor data into database tables are insert queries. The three insert queries are executed at the Node A according to the time period 3. The example FDQs are synthetic. The fault code terminologies used in these FDQs are extracted from [18] and summarized in Table 6.2 .

Our example DMG is shown in Fig. 6.5A.

- QA_1: Insert into EmissionCtrl (ID, CC, EGR, SAI, EVAP, ECB, EPC) VALUES (1, 0425, 0400, 0410, 0440, 0480,0496)
- QA_2: Insert into FuelAirMeter (ID, HO2S, TCWG, CAMSHAFTP, AII, OAT, FPR,

Table 6.2: Sensors Used in Example FDQs

Sensors	Notation
ID	Primary Key
CC	Control Circuit
EGR	Exhaust Gas Recirculation
SAI	Secondary Air Injection
EVAP	Evaporative emission
ECB	Engine Coolant Blower
EPC	Exhaust Pressure Control
HO2S	Heated Oxygen Sensor
TCWG	Turbo Charger Waste Gate
CamshaftP	Camshaft Profile
AII	Air Assisted Injector
OAT	Outside Air Temperature
FPR	Fuel Pressure Regulator
MVAF	Mass or Volume Air Flow
MECTSC	Motor Electronics Coolant Temperature Sensor
HVIC	Hybrid Battery Voltage Isolation Circuit
GTS	Generator Torque Sensor
DMP	Drive Motor Position
GT	Generator Temperature
GPS	Generator Position Sensor

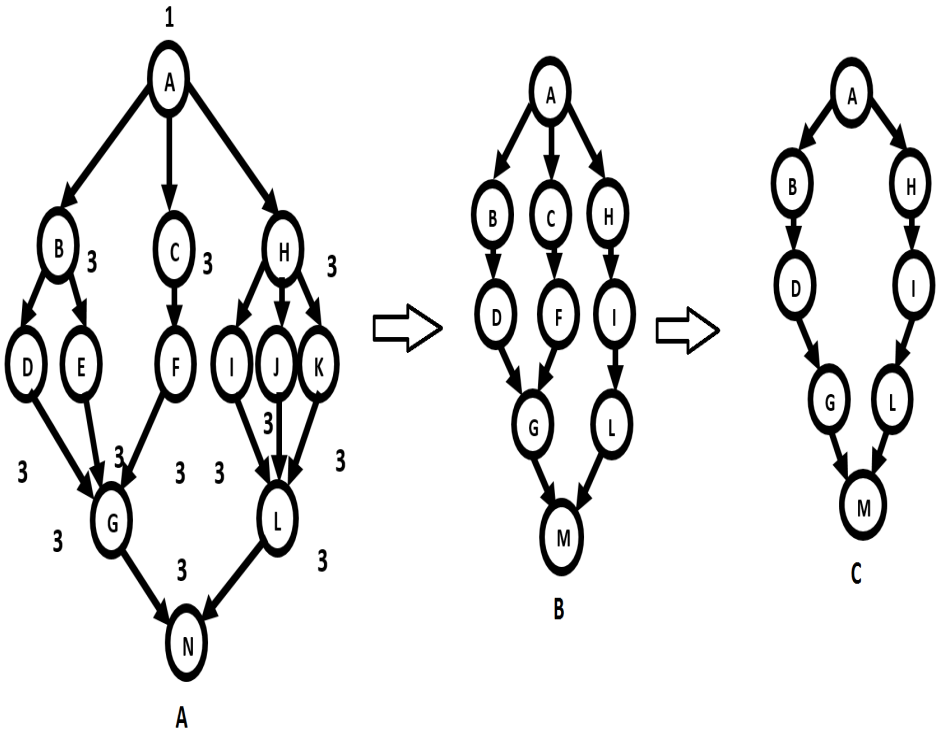


Figure 6.5: Illustrative Example DMG

MVAF) VALUES (1,0030, 0035, 0341, 0065, 0070, 00903, 0294)

- QA_3: Insert into HybridProp (ID, MECTSC, HVIC, GTS, DMP,GT, GPS) VALUES (1, 00, 001, 022, 0306, 036, 0402)

Different sets of values are generated by sensors during the time of 15ms and stored in the corresponding tables. After the execution of FDQs in the parent Node A ends, and the child nodes start executing. There are three children of Node A named as Node B, Node C and Node H. After the execution of FDQs (insert queries) at Node A ends, it (Node A FDQ) sends the required data to all of its child nodes. Data is sent according to the queries present at each node. There are the following queries at these child nodes:

- Q_B: Select EVAP, CC, EGR, SAI from EmissionCtrl where $CC > 0432$. (Results of Q_B query is stored into new table QB).

- Q_C: Select EVAP, ECB, EPC from EmissionCtrl where $SAI > 0418$. (Results of Q_C query is stored into new table QC).

- Q_H: Select * from FuelAirMeter where $HO2S > 0030$ AND $TCWG > 0000$. (Results of Q_H query is stored into new table QH).

At the level two of our DMG the nodes present are Node D, E, F, I, J, K. The parent child relationship of these nodes is as follows:

- Node D, E: These two nodes are child nodes of B.

- Node F: It is the child of Node C.

- Node I, J, K: These three nodes are child nodes of Node H.

The queries present at all these children nodes are as follows:

- Q_D: Select EGR, SAI from QB where $SAI > 0424$

- Q_E: Select CC from QB where $EGR > 0409$

- Q_F: Select EPC, ECB from QC where $EVAP <> 04401$ and $EVAP <> 04402$ and $EVAP <> 04403$

- Q_I: Select HO2S, TCWG, CAMSHAFTP from QH where $TCWG > 0034$.

- Q_J: Select AII, OAT, FPR from QH where $FPR > 00901$

- Q_K: Select MVAF from QH where $MVAF <> 0298$ and $MVAF <> 0299$ (Results of query Q_I, Q_J, Q_K is stored into new table QL).

The nodes present at the third level of DMG are Node G and L. The data required by these nodes are provided by their respective parents. The parent child relationship of these nodes are as follows:

- Node G: This child node has three parents namely, Node D, E, F.

- Node L: This child node also has three parents Node I, J, K.

The join queries present at these child nodes are as follows:

- Q_G: Select QB.CC,QB.SAI,QC.ECB from QB inner join QC on $QB.EVAP = QC.EVAP$ where $ECB <> 0000$

- Q_L: Select HO2S, TCWG, AII, OAT, FPR, MVAF, CAMSHAFTP from QL where $HO2S > 0044$

The node present at the fourth level of our DMG is Node M. The parent child relationship of this node is as follow:

- Node M: It has two parents Node G and Node L.

The join query at Node M is as follow:

- Q_M:Select $QL.HO2S$, QL.TCWG, QL.AII, QG.QBCC, QG.QBSAI from QL inner join QG on $QL.ID = QG.ID$ where $QG.QBCC > 04014$

Graph Optimization Cycle 1

This section describes how the DMG based optimization pruning of the proposed algorithm works (see. Fig. 6.5).

- Step 1: The graph pruning algorithm checks the maximum number of n_i at all levels according to C_1. The maximum number of n_i are present at the second level of the DMG.

- Step 2: At this step the algorithm checks the parents of all neighboring nodes . At the selected level 2 all n_i should be the same. At level 2 Nodes D and E can be merged because they have the same parent node B. Nodes I, J, K can be merged because they have the same parent Node H.

- Step 3: According to C_3, not all children n_i can be deleted. So nodes E, J, K can be deleted only.

- Step 4: According to C_4, the similarity between the SO of prospective merging nodes should be checked. Q_D and Q_E are checked for similar "select" operations. According to queries mentioned in subsection A, Q_D and Q_E have the same SO which is "select" so they can be merged. Similarly Q_I, Q_J and Q_K have the same operation "select" so they can also be merged.

- Step 5: This is the Case 1 of graph pruning as described in the Section 6.2.2. According to C_5 and case 1 of pruning, the data tables for merging $Q(n_i)$ should be the same. According to the queries shown in Section 6.4.1, queries Q_D and Q_E have the same database table from Q_B. Queries Q_I, Q_J and Q_K also have the same database table from Q_H. After the Q_D is merged with Q_E and Q_J, Q_K is merged with Q_I and the resultant query at Q_D and Q_I is as follows:

- Q_D: Select EGR, SAI from QB where $SAI > 0424$ AND $EGR > 0409$

- Q_I: Select HO2S, TCWG, CAMSHAFTP, AII, OAT, FPR, MVAF from QH where $TCWG > 0034$ AND $FPR > 00901$ AND $MVAF <> 0298$ and $MVAF <> 0299$. The new graph generated after optimization cycle 1 is shown in Fig. 6.5B. All the other queries at each node remain the same.

Graph Optimization Cycle 2

After the first cycle of optimization is completed, the algorithm will again check the new generated DMG in order to determine whether additional pruning is still required. If the constraints that are defined in Table. 6.1 are satisfied then the pruning algorithm will again be applied on the DMG. There are two new special cases based on pruning optimization that should be considered at this step.

- Special Case 1: In this case node H cannot be merged with node B and C. For merging the left cluster of nodes is more convenient as the node G has two different parents. In order to keep the data transfer error free the nodes G and L are taken into account and Nodes B, C and H are not considered for merging (see Fig. 6.5B).

- Special Case 2: The algorithm checks for operation C1. The number of nodes present at both the first and second level of the DMG are similar and also maximal. So the algorithm takes the highest level first. It is level 1 and nodes B and C can be considered. After merging node B and C the new query at Q_B becomes:
  Q_B: Select EVAP, CC, EGR, SAI, ECB, EPC from EmissionCtrl where $CC > 0432$ AND $SAI > 0418$. (Results of Q_B query are stored into the new table QB).

Now the algorithm access the second level nodes of the DMG. At this level still C_1 is satisfied so these nodes are merged. The parent node of node F is node C which has already been merged with the node B, so it is also necessary for the node F to be merged with the Node D. This is another special case we considered when the parent node of the child is already merged and the child also has to merge itself with its neighborhood node and with the matching $Q(n_i)$. The data tuples needed by F are from its parent C. These tuples are also transferred to the Node B. $Q(n_i)$ at Node D and F are:

Q_D: Select EGR, SAI from QB where $SAI > 0424$

Q_F: Select EPC, ECB from QC where $EVAP <> 04401$ and $EVAP <> 04402$ and $EVAP <> 04403$.

The data table QC required by query Q_F is now the part of the Node B so the final query at Node D after merging becomes:

Q_D: Select EGR, SAI, EPC, ECB from QB where $SAI > 0424$ and $EVAP <> 04401$ and $EVAP <> 04402$ and $EVAP <> 04403$. The final pruned graph is shown in Fig. 6.5C.

6.6.2 Final Query Execution in DMG

The final FDQs present in the DMG are shown in Fig. 6.5C. Before the execution of the DMG, for each $FDQ(n_i)$ the most optimized QEP with minimum cost is created as described in Section 6.2.4. The worst case QEP is also generated as described in Section 6.2.4. The final $FDQ(n_i)$ at each level of the DMG after pruning is as follows.

Level 1

Q_B: Select EVAP, CC, EGR, SAI, ECB, EPC from EmissionCtrl where $CC > 0432$ AND $SAI > 0418$. (Results of Q_B FDQ are stored into a new table QB)

Q_H: Select * from FuelAirMeter where $HO2S > 0030$ AND $TCWG > 0000$. (Results of Q_H FDQ are stored into new table QH).

Level 2

Q_D: Select EGR, SAI, EPC, ECB from QB where $SAI > 0424$ and $EVAP <> 04401$ and $EVAP <> 04402$ and $EVAP <> 04403$. (Results of Q_D FDQ are stored into a new table QD).

Q_I: Select HO2S, TCWG, CAMSHAFTP, AII, OAT, FPR, MVAF from QH where $TCWG > 0034$ AND $FPR > 00901$ AND $MVAF <> 0298$ and $MVAF <> 0299$. (Results of Q_I FDQ is stored into a new table QI).

Level 3

Earlier the query Q_G was a join query but after merging its parents G has no more joins.

Q_G: Select CC, SAI, ECB from QD where $ECB <> 0000$

Q_L: Select HO2S, TCWG, AII, OAT, FPR, MVAF, CAMSHAFTP from QI where $HO2S > 0044$.

Table 6.3: Resultant Fault Values

HO2S	TCWG	AII	QBCC	QBSAI
0037	03503	0000	040130	04115
0043	0036	06501	040131	04116

Level 4

The final query at Node M has two parents, it is getting the join from the two parent tables *QG* and *QL*.

Q_M:Select $QL.HO2S$, QL.TCWG, QL.AII, QG.QBCC, QG.QBSAI from QL inner join QG on $QL.ID = QG.ID$ where $QG.QBCC > 04014$.

The resultant fault values after the execution of the final query at the last node M are shown in Table 6.3. The last node M is executed after all the pruning and optimization of the DMG is completed at the last phase. The DMG with its WCET is given as an input to the scheduler defined in the system. The scheduler calculates the makespan of the DMG. Table 6.4 and Table 6.5 shows the makespan (as determined by the scheduler) of the example DMG before and after optimization.

6.7 Results

This section elaborates the results based on the graph optimization (graph pruning and merging) and FDQ based optimization (best query execution plans selection) applied to the example DMG. The pruned and optimized DMG is given as an input to the scheduler for the calculation of its makespan.

Fig. 6.6 shows the results when the DMG is considering the case 1. All the nodes which have the DFQs accessing the same database relation are merged. In case of FDQs having select operations the overall optimization applied is based on the selection of the best access path. The best QEP along with the pruned DMG minimized the overall makespan of the DMG by up to 40% when the proposed technique is applied. Fig. 6.7 shows the results

Table 6.4: Data Statistics for makespan of DMG before Optimization (seconds)

Nodes	Query	$WCET(Q(n_i))$
A	$QA_{1,2,3}$	0.066
B	QB	0.036
C	QC	0.034
H	QH	0.033
D	QD	0.031
E	QE	0.029
F	QF	0.039
I	QI	0.032
J	QJ	0.031
K	QH	0.039
G	QG	0.056
L	QL	0.032
M	QM	0.058
Makespan		4.16

Table 6.5: Data Statistics for makespan of DMG after Optimization (seconds)

Nodes	Query	$WCET(Q(n_i))$
A	$QA_{1,2,3}$	0.066
B	QB	0.027
H	QH	0.026
D	QD	0.025
I	QI	0.024
G	QG	0.036
L	QL	0.026
M	QM	0.038
Makespan		2.04

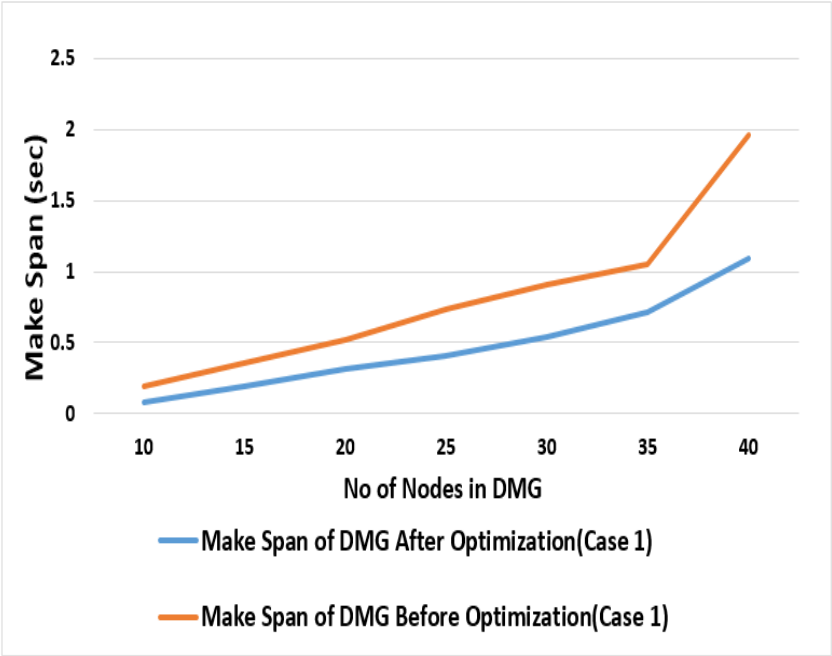


Figure 6.6: Makespan of DMG Case 1

when the DMG is considering the case 2. FDQs in this case have join operations which require more optimization, as the best join orders also have to be selected. In this case a maximum of three join operations is considered. Therefore at each level of the DMG a maximum of three nodes can merged along with their FDQs. In the context of this scenario it is evident that when there are graphs with larger sizes which require more pruning and more query optimization then the proposed technique works more effectively.

Fig. 6.8 shows the results when the DMG considers both cases and the DMG has both types of FDQs. There are different numbers of select and join FDQs within the DMG. It is evident from the results that in case where we have higher numbers of FDQs with join operations and lesser numbers of FDQs with select operations, graph pruning and query optimization algorithm is more effective.

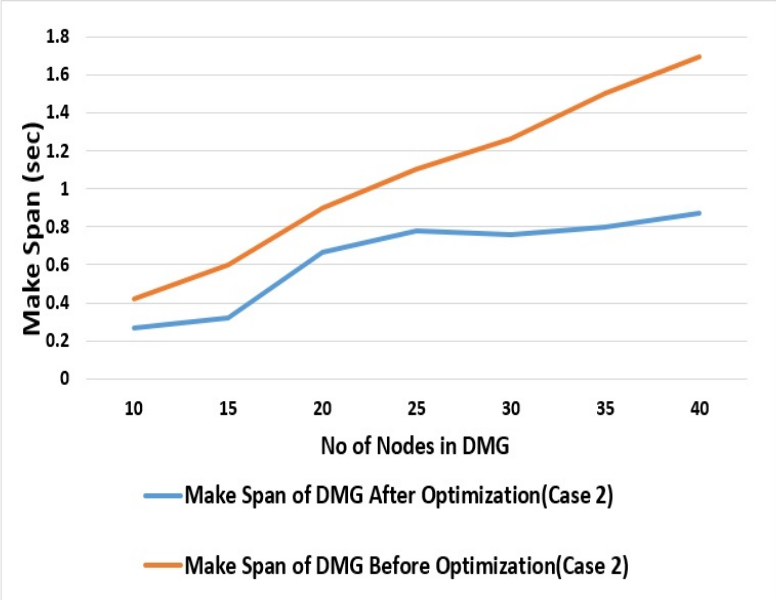


Figure 6.7: Makespan of DMG Case 2

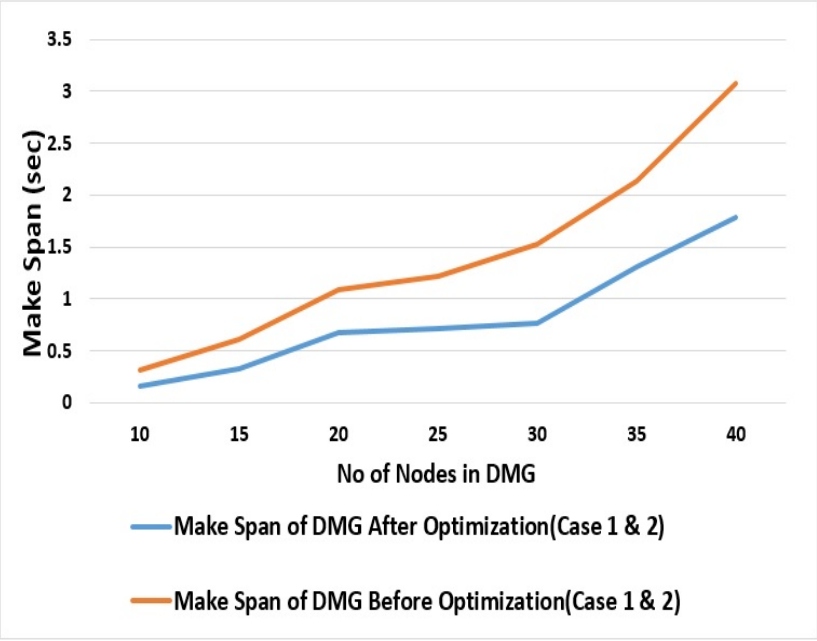


Figure 6.8: Makespan of DMG Case 1 & Case 2

CHAPTER 7
MINIMIZING THE MAKESPAN OF DMGS USING QUERY AWARE PARTITIONING

This chapter describes the minimization of the makespan of our proposed DMG by using the technique of query aware partitioning.

7.1 Optimization Algorithm

The pseudo-code representation of our proposed technique is shown in Algorithm 3.

The steps followed for the implementation of our technique are shown in Fig. 7.1. The basic implementations details for this algorithm are as follow: Algorithm 3 minimizes the overall makespan of the DMG, by minimizing the WCET of each $FDQ(n_i)$. This goal is achieved by employing a query aware partition pruning algorithm on the created FDQs [126]. The second objective is the reduction of resource utilization. This objective is achieved by minimizing the overall data (and thus communication cost) transferred among the edges of DMG. This goal is achieved by introducing two new concepts in the proposed application model presented in Fig. 4.2. These features are history intervals (HI) and the Skip value (S). The DMG with these two new inputs is shown in Fig. 7.2. For select queries the Per Table Query Aware Partitioning (PTP) is applied and for join queries, the Join Aware Query Partitioning (JAP) is applied [127]. Both techniques have different results as they are applied on different types of FDQs.

Algorithm 3 Query Optimization Algorithm (QAA)

Input:DMG
Output:Optimized DMG, WCET, W_i

```
1: while the last node of DMG is not reached do
2:     while node has not completed its all executions do
3:         if SQL operation of query is "Select" then
4:             Apply PTP to data table in query
5:             Calculate W_i for data table after partitioning
6:             Generate QEP
7:         end if
8:         if SQL operation of query is "Join" then
9:             Apply JAP to table in query
10:            Calculate W_i for data table after partitioning
11:            for all JO in a query do
12:                Select JO with minimum cost
13:            end for
14:            Generate QEP
15:        end if
16:        Calculate WCET for QEP
17:        Read history interval for current node
18:        if there is a skip value S then
19:            Do not send result partition to child node
20:        else
21:            Send result partition to the child node
22:        end if
23:    end while
24: end while
```

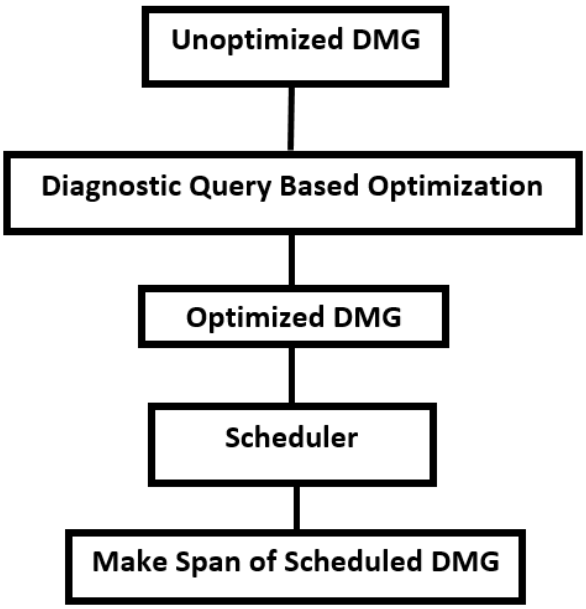


Figure 7.1: Steps in Proposed Technique

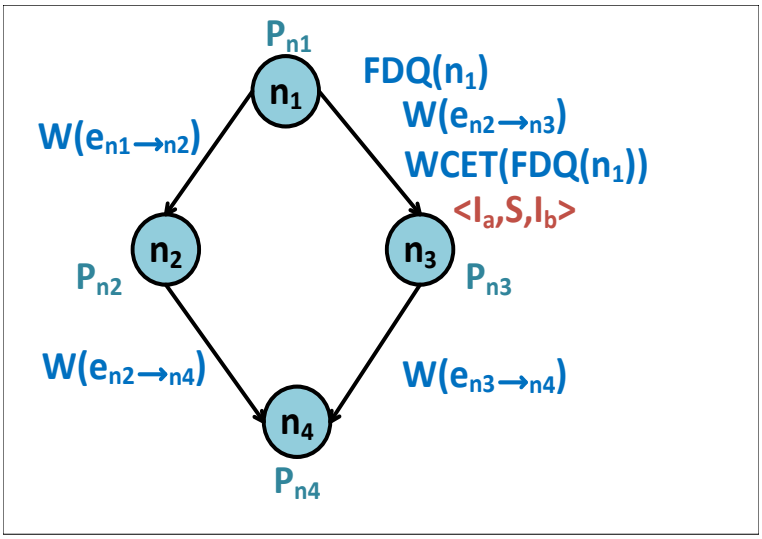


Figure 7.2: DMG with History Interval and Skip Factor

7.1.1 Basic Optimization Rules

There are certain sets of rules applied by modern bottom-up optimizers for query optimization [59]. These optimization rules are as follow.

1. The best access method is determined for each database relation in $FDQ(n_i)$ by employing the table scan or index scan. The selected access path should have less cost in terms of processing time.

2. The best join path is found for each join query. These join paths can be merge join, hash join and equi-join. The join path with minimum cost in terms of execution time is selected.

7.1.2 Per Table Optimization for Select Queries

For select queries, the proposed query optimizer applies the technique of PTP. This type of pruning deletes all those table partitions which are not required by the $FDQ(n_i)$. This optimization minimizes the memory requirements of data storage along with the unnecessary overheads by deleting unused data tuples [128]. An example FDQ, Q_1 is considered for the explanation of this concept.

Q_1**: Select A from R where** $A > 10$

The child fragments in Fig. 7.3A which are black in color are pruned after applying PTP optimization. After PTP the QEP for each $FDQ(n_i)$ is generated. The QEPs before optimization and after optimization (less WCET) are shown in Fig. 7.3C and Fig. 7.3B. Hence the overall WCET is reduced by almost 50% after the PTP is applied.

7.1.3 Join Aware Partition Optimization for Join Queries

For join queries, JAP and pruning are applied (see Algorithm 3, Lines 8-10). After this the proposed query optimizer selects the best join order with minimum cost (see Algorithm 3, Lines 11-13). An example query is represented as follows:

Q_2: Select * from A, B, C where A.u=B.u AND B.v=C.v AND $A.u > 10$

The JAP for query Q_2 is shown in Fig. 7.4A. The partitions in black are pruned. In the next step our optimizer finds the best join order. The join operation in case of Q_2 is comprised of three tables $(A \bowtie B) \bowtie C$. The Sp for the number of join orders is bigger

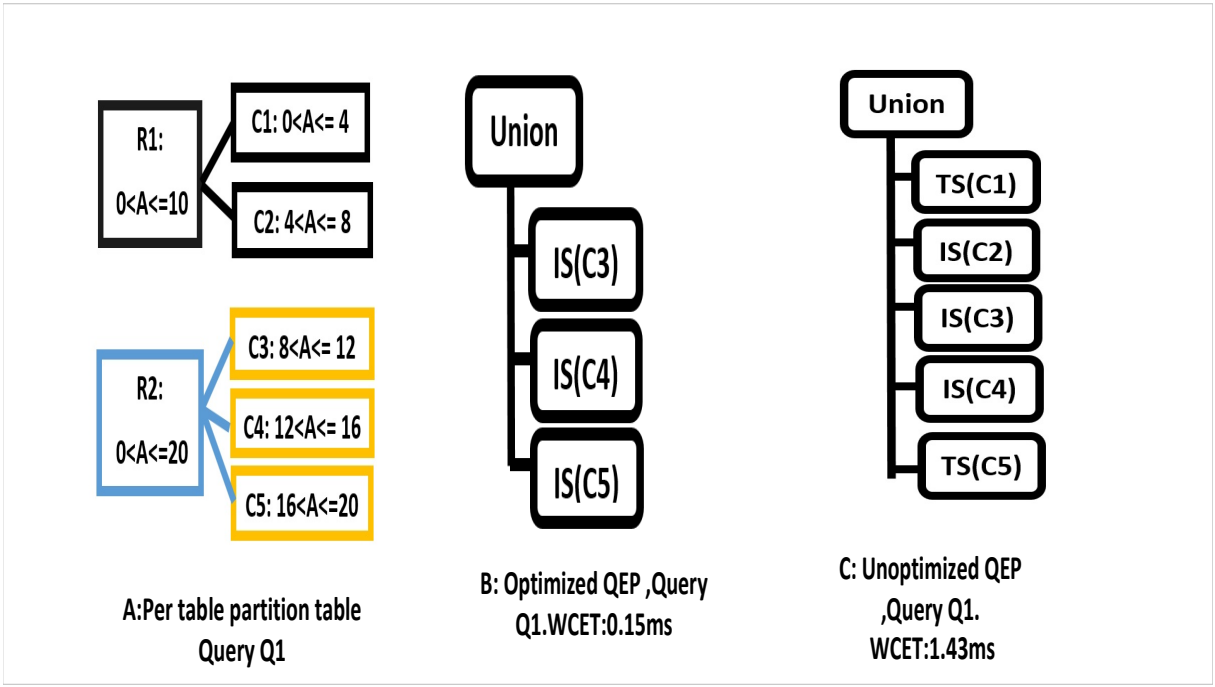


Figure 7.3: A:PTP, B:Optimized QEP, C:Unoptimized QEP for Query Q_1

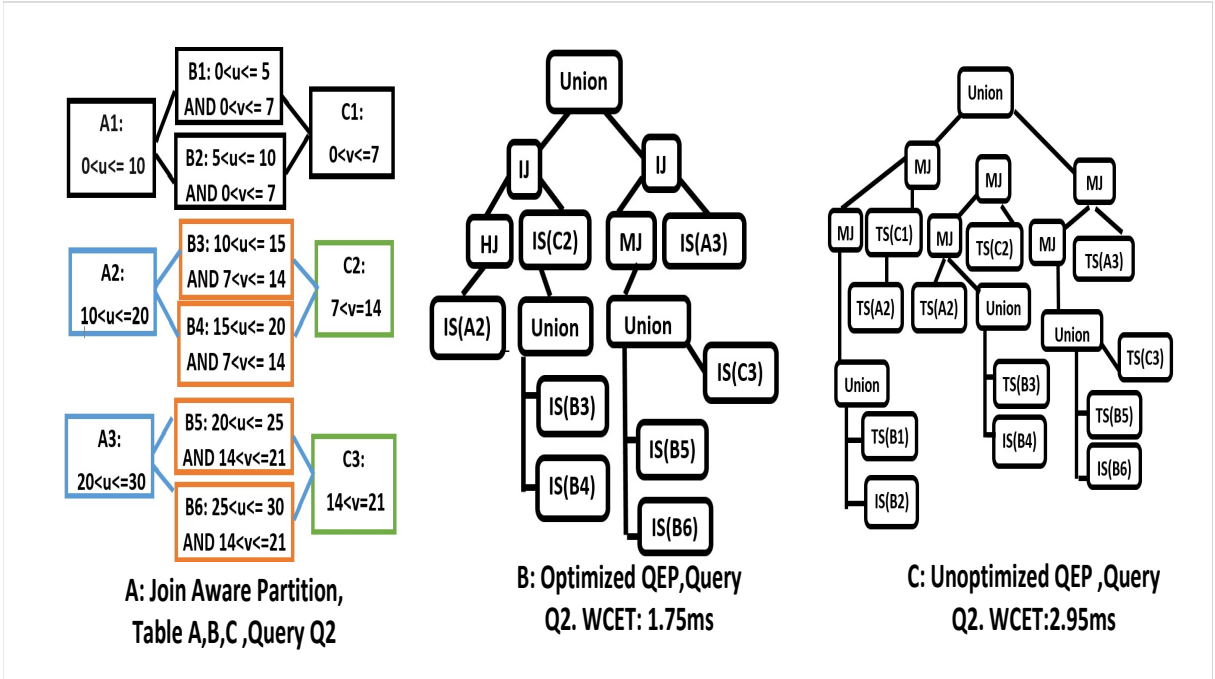


Figure 7.4: A:Join Partition, B:Optimized QEP, C:Unoptimized QEP for Query Q_2

because of the data fragments A2, B3, B4, C2, A3, B5, B6, C3. In order to minimize the Sp for join orders the clustering technique is applied. During clustering we select only those child partitions of the query table which have the same parents, so according to Fig. 7.4A the join orders considered for the search space are $A2 \bowtie (B3 \bowtie B4)$ and $A3 \bowtie (B5 \bowtie B6)$.

Now optimizer selects the best join path for these joins. The best join path for $A2 \bowtie (B3 \cup B4)$ is merge join and for $A3 \bowtie (B5 \cup B6)$ it is hash join as both of these paths have less cost. Now for each join order, the child fragments are considered. Proposed optimizer selects the best join order. These child join orders for $(A \bowtie B) \bowtie C$ are considered first and they become $(A2 \bowtie (B3 \cup B4)) \bowtie C2$ and $(A3 \bowtie (B5 \cup B6)) \bowtie C3$. After that, the child join order for $(B \bowtie C) \bowtie A$ is considered. The child join order become $((B3 \cup B4) \bowtie C2) \bowtie A2$ and $((B4 \cup B5) \bowtie C3) \bowtie A3$. Now the optimizer determines which order is best depending upon the minimum cost calculation and generates the QEP for that. The QEP before and after partition pruning is shown in Fig. 7.4C and Fig. 7.4B.

7.2 Calculation of WCET

We estimate the WCET, i.e the maximum time that a FDQ can take for its execution [129]. For determining the WCET of each FDQ, a QEP is generated. The time of each QEP is measured by considering all the possible sets of data tuples that a FDQ can process. Therefore the WCET is calculated by reading the overall statistics generated by the QEP. Divergent cost related factors including I/O cost and CPU cost are analyzed and estimated from the QEP generated by PSQL [116].

7.3 Illustrative Example

This section elaborates the illustrative example on which the proposed algorithm is implemented and tested.

7.3.1 Implementation of History Interval and Skip Factor

There are three different queries in the proposed context including insert, select and join. The major objective is to minimize the overall WCET of select and join queries by applying the proposed optimization. Fig. 7.5A represents the example DMG. According to Fig. 7.5A the following history intervals are considered for each n_c:

- $(A \rightarrow B)$: $< 1,0,4 >$ means that Node B requires all the data from the Node A as the skip value S=0 while start execution I_a=1 and end execution I_b=4. S=0 is represented by blue lines from $(A_1 \rightarrow B_1)$, $(A_2 \rightarrow B_1)$, $(A_3 \rightarrow B_3)$ and $(A_4 \rightarrow B_2)$ in Fig. 7.5B.

- $(B \rightarrow C)$: $< 1,2,2 >$ means that start execution I_a=1 and end execution I_b=2 with the skip value S=2. So Node C requires the data from the first execution of Node B as shown by the blue line from $B_1 \rightarrow C_1$, and skips the data from the second execution of Node B as shown by the red line from $B_2 \rightarrow C_1$ in Fig. 7.5B.

- $(A \rightarrow C) < 3,0,4 >$ means that the start execution I_a=3 and the end execution I_b=4 with the skip value Sk=0. So Node C requires the data from the third and forth executions of Node A as shown by the blue line $A_3 \rightarrow C_1$ and $A_4 \rightarrow C_1$. The data from the first and second execution of Node A is skipped as denoted by the red lines from $A_1 \rightarrow C_1$ and $A_2 \rightarrow C_1$ in Fig. 7.5B.

7.3.2 Number of Executions on Basis of Skip Factor

Fig. 7.5B represents the mapping to the hyper period based on Fig. 7.5A. It represents the number of executions for each node, along with the data that is required by the child node (n_c) from its parent node (n_p) in a specified execution sequence. The lines drawn in red show the skipped executions of a parent node from which the child node does not require data. The number of executions on the basis of skip factor are described in next section:

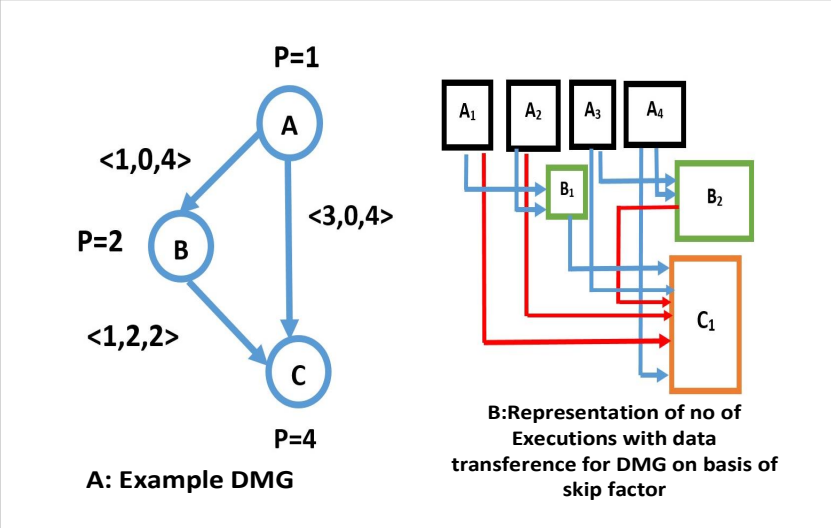


Figure 7.5: A:Example DMG, B:No of Executions with Skip Factor

- Node A: It has four executions according to its time period P=1 denoted by (A_1, A_2, A_3, A_4). It repeats its execution every second.
- Node B: It has two executions according to its time period P=2 denoted by (B_1, B_2). It starts its first iteration after it receives data from the first and second iteration of Node A according to the history interval $< 1,0,4 >$ and starts its second execution after it receives the data from the third and forth execution of Node A according to the history interval $< 1,0,4 >$.
- Node C: It has one execution according to its time period P=4 denoted by C_1. Node C starts its single iteration after it receives its data from the first iteration of B according to the history interval $< 1,2,2 >$, and the third and fourth iteration of A according to the HI $< 3,0,4 >$.

7.3.3 Query Optimization

Node A is the root node which is receiving all the input data from the sensors. FDQs having insert operations store the data into the database. Suppose we have the following FDQs for populating the database:

- QI_1:*Insert into Oxygen where OxygenRatio is between 0.1 and 2.9 and* CO_2*Ratio between 0.1 and 1.8.*
- QI_2*:Insert into Water where WaterTemp is between 0.1 and 2.3.*
- QI_3*:Insert into Engine where EngineTemp is between 0.2 and 1.9.*
- QI_4*:Insert into Fuel where FuelTemp is between 0.5 and 2.7.*

Execution of Node A

Now we consider the queries that are present at Node A of the DMG. We can see that we have P=1 as shown in Fig. 7.5A. We consider that we have four select FDQs at Node A during different executions. The FDQs designed for node A are as follows:

- QA_1:*Select * from Oxygen where OxygenRatio* > 0.1 *and OxygenRatio* < 0.3
- QA_2*:Select * from Water where WaterTemp* > 0.1 *and WaterTemp* < 0.4.
- QA_3*:Select * from Engine where EngineTemp* > 0.3 *and EngineTemp* < 0.5
- QA_4*:Select * from Fuel where FuelTemp* > 0.5 *and FuelTemp* < 1.1

After the execution of four queries the PTP and QEP are shown in Fig. 7.6. The ratio of data that is pruned in QA_1 is 40%, 25% for QA_2, 28% for QA_3 and 42% for QA_4.

First Execution of Node B

- QB_1:Select Oxygen.OxygenRatio, Water.WaterTemp from Oxygen inner join Water ON Oxygen.ID=Water.ID where Oxygen.OxygenRatio$>$ 0.6 and Water.WaterTemp > 0.5

Node B requires the data from the $< 1,2 >$ execution of Node A to complete its first execution. According to Fig.7.5B, Node A has already transferred its data to the Node B.

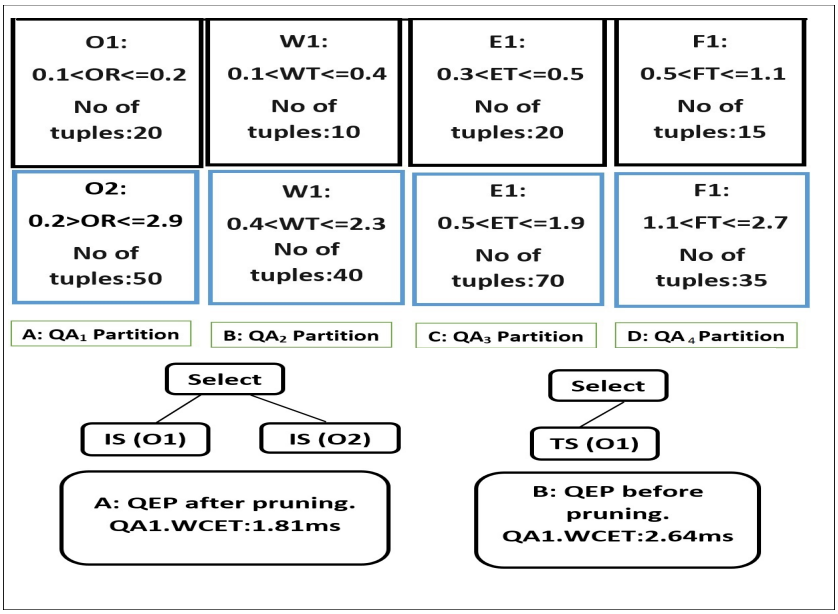


Figure 7.6: PTP for QA_1, QA_2, QA_3, QA_4

Node B has two executions according to its period. After the data is transferred from the Node A, now the FDQ at Node B is QB_1. Now our optimizer applies the JAP to the QB_1. The JAP applied by the optimizer is shown in Fig. 7.7A. Therefore the partitions O3 and W5 are pruned. The QEP applied before and after pruning are shown in Fig. 7.7C and Fig. 7.7B.

Second Execution of Node B

- QB_2:Select EngineTemp,FuelTemp FROM Engine INNER JOIN Fuel On Engine.ID=Fuel.ID where Engine.EngineTemp$>$ 0.6 AND Fuel.FuelTemp $>$ 1.3

After the first execution of Node B, it transfers its data to the Node C. The second execution of the Node B requires the data from the $< 3,4 >$ execution of the Node A which has already been transferred by Node A. After the data transfer from the Node A is completed, now the query at the Node B is QB_2. The JAP applied to the Node B at its second execution is shown in Fig. 7.8A and the QEP before and after pruning is shown in Fig. 7.8C and Fig. 7.8B.

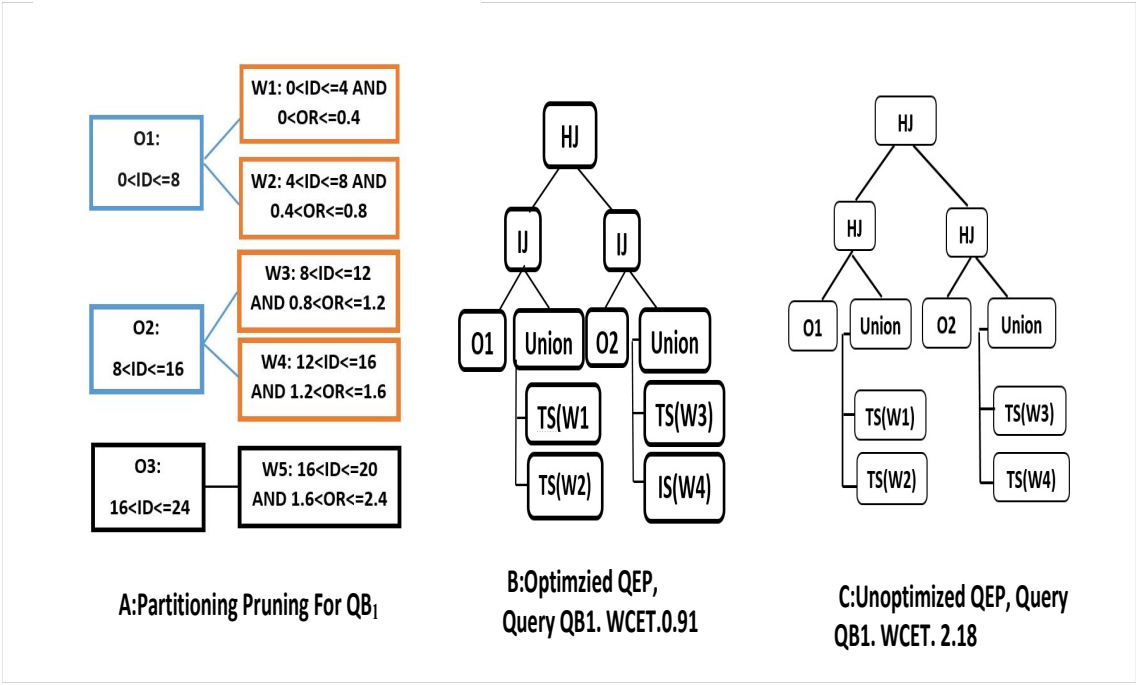


Figure 7.7: A: JAP, B:WCET of Optimized QEP, C:WCET of Unoptimized QEP for QB_1

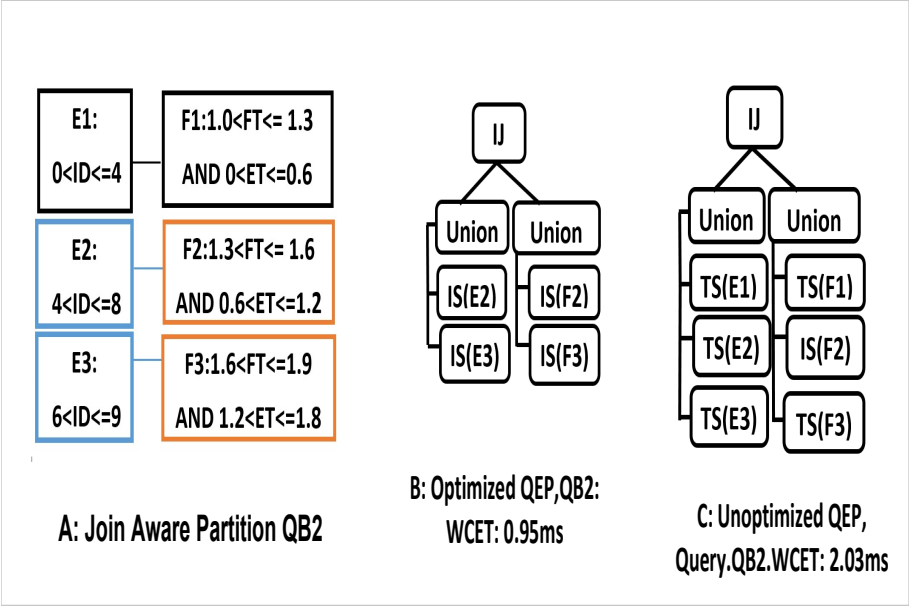


Figure 7.8: A:JAP, B:WCET of Optimized QEP, C:WCET of Unoptimized QEP for QB_2

Table 7.1: Resultant Values Extracted from FDQ QC_1

OxygenRatio	EngineTemp	FuelTemp
1.8	1.5	2.1
2.5	1.5	2.2

Execution of Node C

Now the last Node C of the DMG executes. Node C requires data from two nodes: Node A and Node B. All of this data has already been sent to the Node C when the executions of Node A and Node B end.

- Node A: Node C requires the data from the third and fourth iteration of Node A.
- Node B:Node C requires data from the first execution of Node B.

The final query at Node C is QC_1. According to the period of Node C (P=4 according to Fig.7.5A), it has to execute only one time. Now the relations that are present at the Node C are the join table Oxywater, Engine and Fuel. The JAP and QEP before and after optimization for QC_1 are shown in Fig. 7.9A, 7.9B and 7.9C.

- QC_1:Select OxyWater.OxygenRatio,Engine.EngineTemp, Fuel.FuelTemp FROM OxyWater INNER JOIN Fuel ON OxyWater.ID=Fuel.ID INNER JOIN Engine ON Fuel.ID=Engine.ID where OxyWater.OxygenRatio$>$ 0.9 AND Engine.EngineTemp $>$ 1.4

After the complete Execution of Node C

After the final query at Node C executes the tuples extracted from the FDQs are shown in Table. 7.1. These value are compared with the threshold values stored in database and fault is detected.

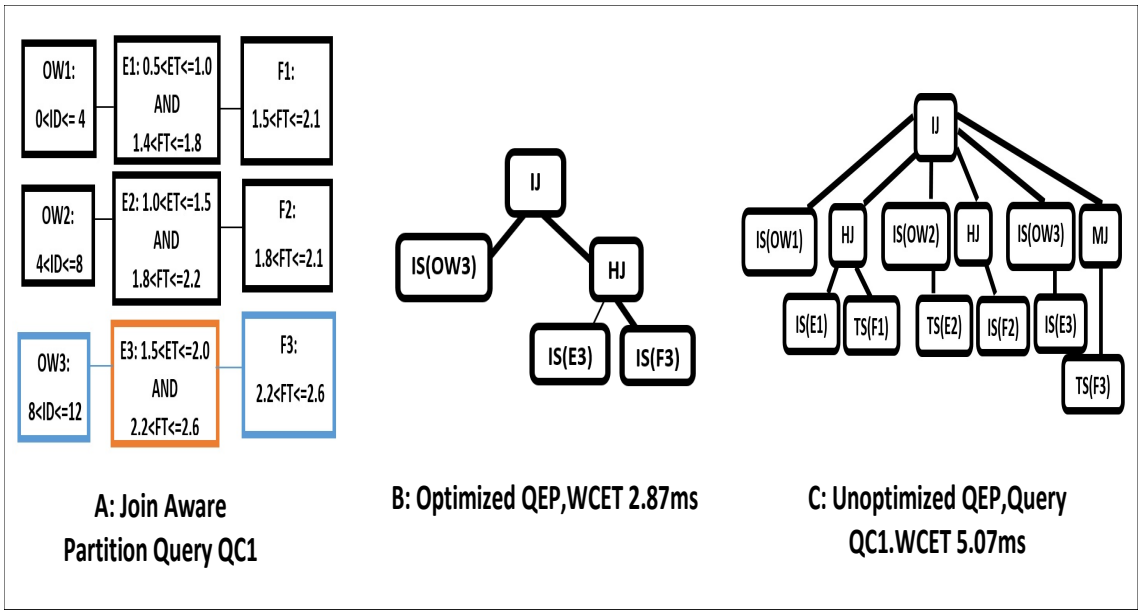


Figure 7.9: A: JAP, B:WCET of Optimized QEP, C:WCET of Unoptimized QEP for QC_1

Table 7.2: W_i with Different Values (KBs)

Parent $\rightarrow$ *child*	With Skip value	With skip value and partition	No (skip value and partition)
$A \rightarrow B$	88	58.88	88
$A \rightarrow C$	48	31	88
$B \rightarrow C$	47.8	41.3	58.88
W_i		Optimize: 131.18	Unoptimized: 234.88

7.3.4 Results for Example DMG

The Table 7.2 represents the amount of data transferred within the the nodes of DMG before and after the partition and skip value is applied. The Amount of Data (Amt Data W_i) is shown With Skip value and With Skip value and Partitioning and Without Skip value and Partitioning.

The resultant values in Table. 7.2 shows that the amount of data after applying S and P is decreased by 30% to 45% depending upon the type of FDQ. Table 7.3 shows that the WCET is decreased by almost 40% after applying the optimization.

After the application of the history interval and the query aware partition, the optimized

Table 7.3: WCET for Optimized QEP (OQEP) and Unoptimized QEP (UQEP)

FDQs at each Node	WCET(OQEP)	WCET(UQEP)
QA_1, QA_2, QA_3, QA_4	3.89	6.92
QB_1	0.91	2.18
QB_2	0.95	2.03
QC_1	2.87	5.07

Table 7.4: Makespan for Optimized DMG and Unoptimized DMG

DMG	Make Span (Secs)
Optimized DMG	6.87
Unoptimized DMG	11.04

DMG with the WCET of each query node (cf. Table. 7.3) and the amount of data (W_i) along each edge of the DMG (cf. Table 7.2) is given to scheduler for finding the makespan. The makespan of an example DMG in Fig. 7.5A is shown in Table 7.4.

The resultant values in Table 7.4 shows that the overall makespan of the DMG is reduced up to 39% after the optimization is applied.

7.4 Results

This section elaborates the results based on the architectural model shown in Fig. 4.1. The proposed optimization technique was implemented in Java and tested with a PSQL server. For the evaluation of the proposed algorithm, the implementation of the optimizer is divided into three different categories.

1. Optimizer considers the DMG with only select queries and applies PTP to it.

2. Optimizer considers the DMG with only join queries and applies JAP to it.

3. Optimizer considers the DMG with both select and join queries and applies PTP and JAP both according to the type of a query.

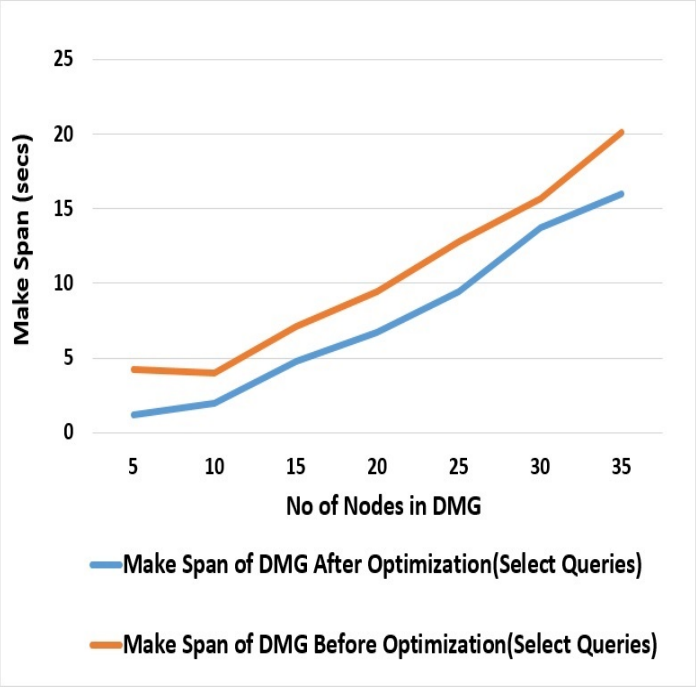


Figure 7.10: Comparison of Makespan of DMG (select FDQs)

7.4.1 Description of Results

This section presents the results based on the query optimization applied to the DMG. The objectives achieved in our context are the (i) minimization of makespan of DMGs for meeting the strict timing constraints given by the system and (ii) minimizing the resource consumption (i.e.,CPU and bandwidth). The makspan of the DMG is calculated by providing the WCET of FDQs and W_i (weight of the edge(to our scheduler designed in [122].

Fig.7.10 presents the makespan calculation of the DMG with FDQs having only ”select” operations. For this type of DMG the PTP is applied. Fig.7.11 presents the makespan calculation of the DMG with FDQs having only ”join queries”. We have considered three relations in case of join queries. For join queries the JAP is applied. Fig. 7.12 presents the makespan calculation of the DMG having FDQs with both ”select” and ”join” operations. Select queries based on the ranges are considered and join queries based on a maximum of

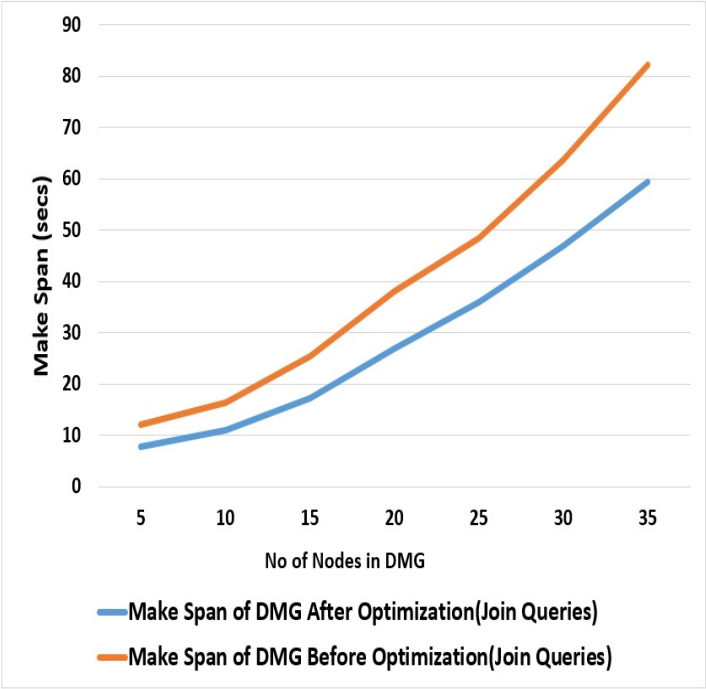


Figure 7.11: Comparison of Makespan of DMG (join FDQs)

three joins are considered.

According to Fig. 7.10, Fig. 7.11 and Fig. 7.12, there is a significant reduction in the overall makespan of the DMG, which is the major objective of our work. This is a significant result and can be exploited in the case where we have safety-critical systems with stringent timing constraints. Fig. 7.13 shows the CPU consumption of DMG before and after the optimization. The CPU consumption is calculated by $U = PT/C$ where U= Consumption, PT= Total Execution Time (ET) taken by each DMG node, C= Capacity of CPU, which is ET+IT (Idle time). Results in this context show a reduction of CPU utilization up to 30% after optimization.

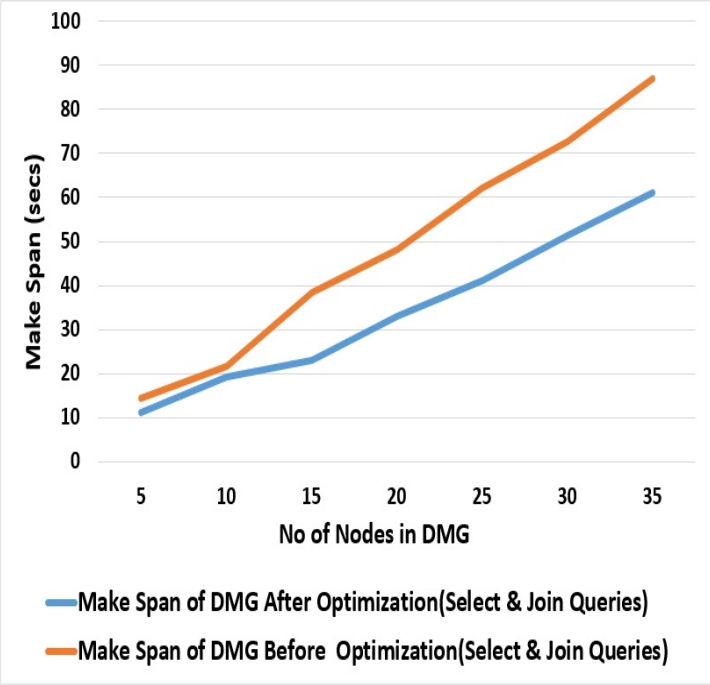


Figure 7.12: Comparison of Makespan of DMG (select and join FDQs)

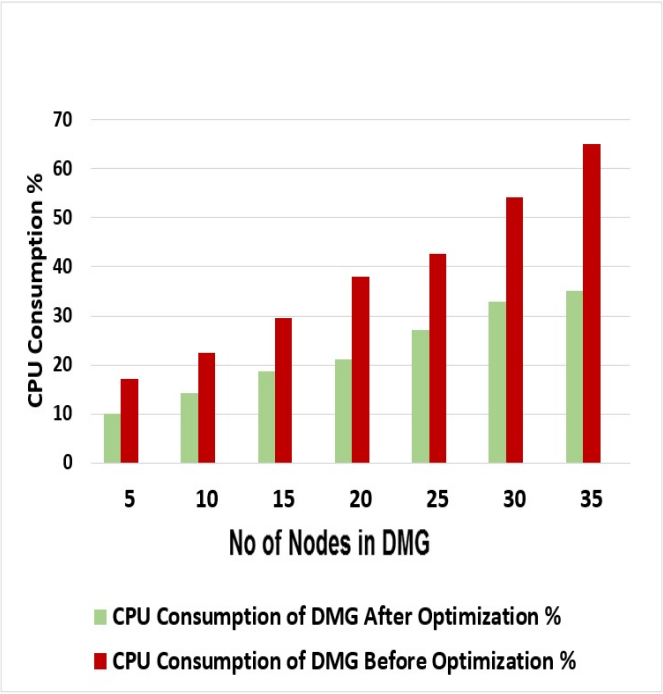


Figure 7.13: Comparison of CPU Consumption Before and After Optimization)

CHAPTER 8

MINIMIZING THE MAKESPAN OF DMGS USING GENETIC ALGORITHM

This chapter describes the GA based technique, which is used for minimizing the makespan of our DMGs. Details about the optimization of FDQs along with the example and results are also presented in this context. The steps followed for the implementation of this technique are shown in Fig 8.1.

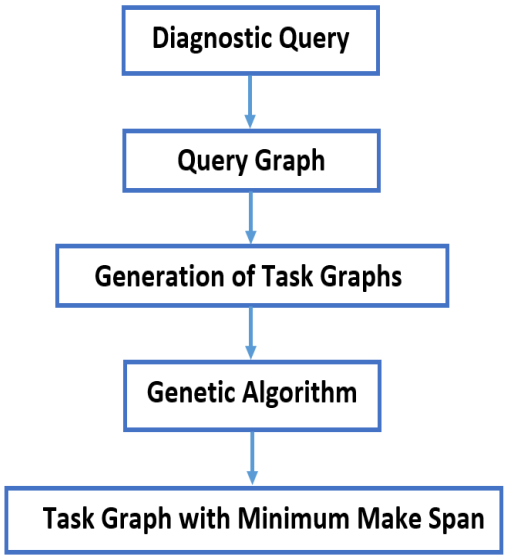


Figure 8.1: Steps in Proposed Algorithm

8.1 Optimization Algorithm

This section describes the pesudo-code representation of our algorithm presented in Algorithm 4.

Algorithm 4 Task Graph Optimization Algorithm (TGO)

Input:Diagnostic Query (DQ)
Output:Optimized Task Graph (TG), Makespan of TG

```
1: Convert DQ into Relational Algebra (RA) representation.
2: Convert RA of DQ into Query Tree (QT)
3: QT can be left depth tree or bushy tree
4: Generate different Task Graphs (TG) with different combinations of nodes from QT.
5: Add all these TG into Search Space (SP) of Genetic Algorithm (GA)
6: Create initial population based on these TG for GA.
7: while the TG with minimum Make Span is not found do
8:     for all TG in initial population do
9:         Calculate fitness function (makespan)
10:    end for
11:    Select TG with minimum makespan
12:    Pass them to next generation
13:    Apply mutation on selected solutions (TG)
14:    Send them to next generation.
15:    Exit when TG with minimum makespan is found
16: end while
```

8.2 Important Components of Technique

This section elaborates all the important components which are defined for the implementation of our technique. The details about the Genetic Algorithm (GA) is also mentioned in this context.

8.2.1 Fault Diagnostic Query (FDQ)

In this work, we have considered only one FDQ for the simplification of our scenario and understanding of the reader. Designed FDQ in this context is quite complex as it comprised of multiple join operations.

8.2.2 Query Tree

The second step in the proposed algorithm is to convert the FDQ into a query tree. The query tree is the algebraic representation of our FDQ. The internal nodes of the query tree represent the relational algebra operations while the leaves represent the database relations

present in the FDQ. We have considered two types of trees, including (i). Left Deep Tree (LDT) and (ii). Bushy Tree (BT) (Fig. 8.3).

8.2.3 Task Graph

For each query tree, different task graphs are generated. For example, in the example FDQ $WCET(T_5)$ represents the WCET of task node T_5 and $W(e_4)$ represents the weight of the edge e_4. $W(e_4)$ represents the amount of data that is being transferred from parent node (T_4) to child node (T_5) (see Fig. 8.4B). It is assumed that each edge has a weight that is $W(e) = 1$. Whenever the data item of size S is sent from n_p to n_c, the overall cost of the edge becomes $S.W(e_i)$. Each task graph is considered as a solution for the search space in the GA. Each task graph is comprised of a different number of leaf nodes from one query tree. The type of these task graphs depends on the type of query tree from which they are created. These task graphs are either LDT based TG or BT based task graphs.

8.2.4 Genetic Algorithm (GA)

The last step in the proposed solution is to apply the GA so that the task graph with minimum makespan is found. The next section describes the necessary steps for the implementation of the GA.

Initial Population

Different task graphs (i.e.,individual solutions) that are created on the basis of LDT or BT (Query tree) are considered as the initial population for the creation of the search space (see Algorithm 4: Lines 5-6). Binary strings are used for the chromosome representation of solutions. A binary value 1 is assigned to each SQL operation within the query tree if it is considered as a separate operation in the TG. And a binary value 0 is assigned to a particular SQL operation in a query tree if it is considered in combination with its child node (each node contains an SQL operation). For example if the parent SQL operation is

combined with the child SQL operation, then two joins are combined together in the form of one SQL query. Then the parent SQL operation is assigned 1 and child SQL operation is assigned 0 as a binary representation of the solution.

Fitness Function

The calculation of the fitness function is a significant step in the GA. The WCET of each graph node along with the $W(e_i)$ for each TG is given as an input to the scheduler for calculating the maekspan. The calculated makespan of each task graph is considered as the value of the fitness function. The calculation of the fitness function is performed by our scheduler designed in [122]. The scheduler is invoked whenever the fitness function of the solution in the search space has to be calculated.

Selection

This step of the GA selects the fittest individuals and passes them to the next generation. For each solution, makespan is calculated. The solutions with minimum makespan are selected for the next generations.

Mutation

Cross over for the solutions is not possible. If cross over is performed over two TGs then there is a chance that the final solution becomes wrong due to the change of the node's position. If the mutation between two solutions is not able to create a new solution then GA will quit its execution.

Termination

The proposed algorithm terminates when the GA keeps finding similar values of makespan for each solution. After the certain number of generations, the makespan of task graphs start converging. In this case, the GA will terminate its execution.

8.3 Determining the Worst Case Execution Time

The GA depends on the calculation of the makespan for each task graph. For calculating the makespan the WCET of each node in the task graph is required. Each node of the task graph comprises different SQL operations based on the FDQ. These FDQs are executed over the vehicle database created in PSQL. For finding the WCET of each node in a TG, the SQL operation in this graph node is separated from the main FDQ and converted into a sub query. These sub queries are measured against all the combinations of data extracted from the database. For our evaluation we have three thousand FDQs that are run over the designed database for calculating the WCET of different task graphs.

8.4 Example

This sections describes the illustrative example for our proposed algorithm.

8.4.1 Fault Diagnostic Query

As described in Section 4.2 FDQs are represented in the SQL format. The execution of these FDQs is performed over that data stored in a PSQL server. This database comprises data that is derived from ehicle sensors. The example FDQ is denoted by Q_D.

Q_D: Select Oxygen.OxygenRatio,Water.WaterTemp,Oil.Oillimit, Engine.EngineID from Oxygen, Oil, Water,Engine,Fuel where Fuel.WaterID=Water.WaterID and Water.WaterID =Engine.EngineID and Oil.OilID=Engine.OilID and Oxygen.OxygenID=Oil.OxygenID

The query Q_D is a join query and comprised of four join operations along with five different relations from database. This FDQ is complex because it contains various joins.

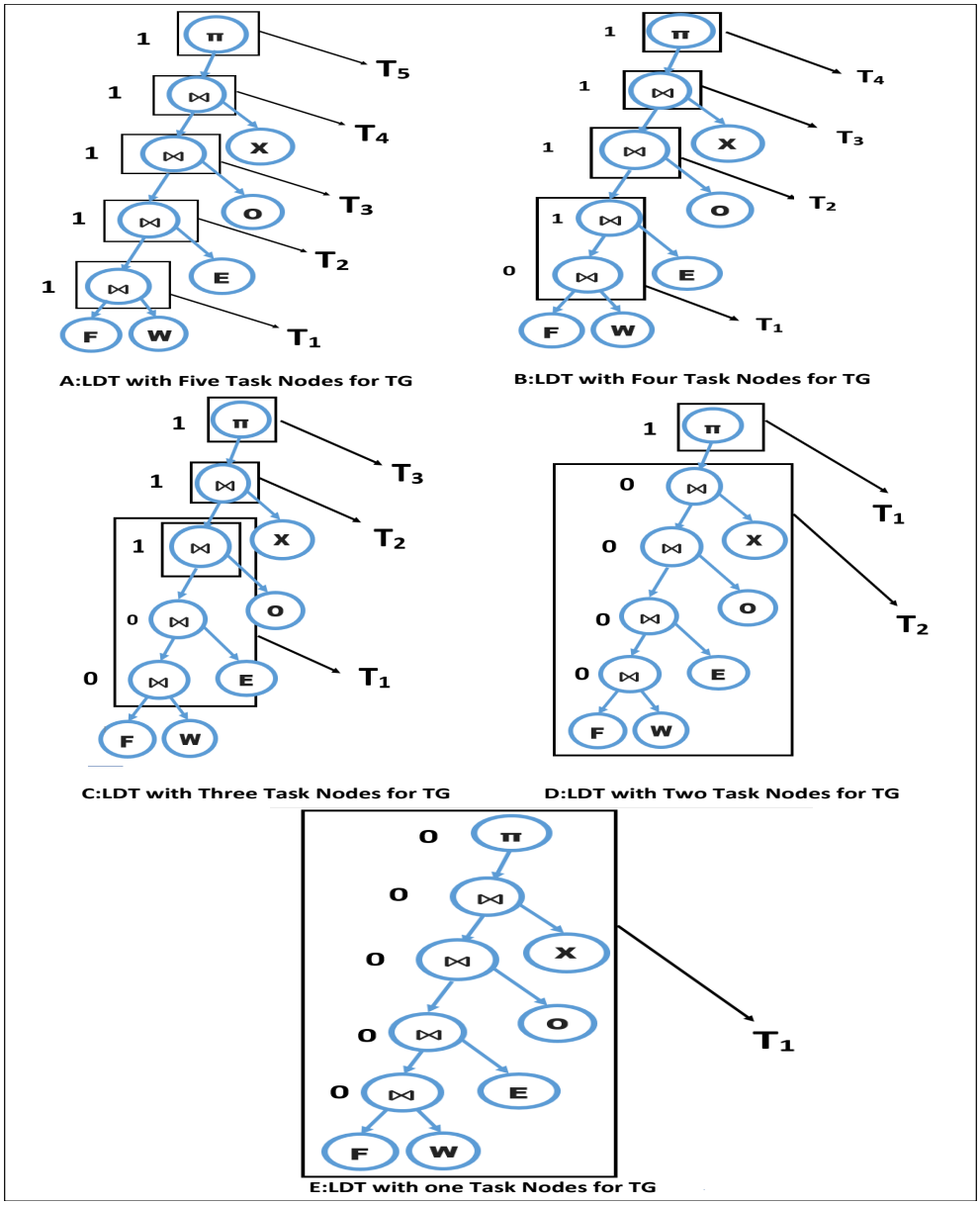


Figure 8.2: LDT based TG with Binary Representation

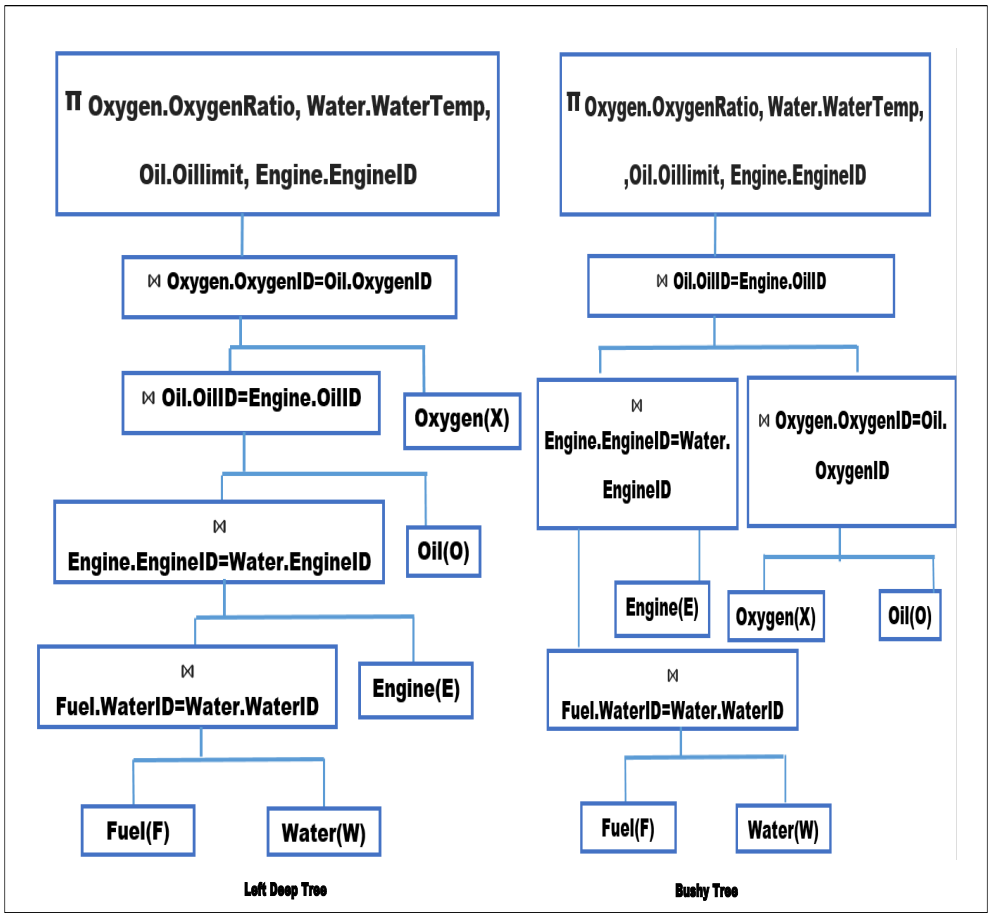


Figure 8.3: Query Tree Left Deep Tree and Bushy Tree

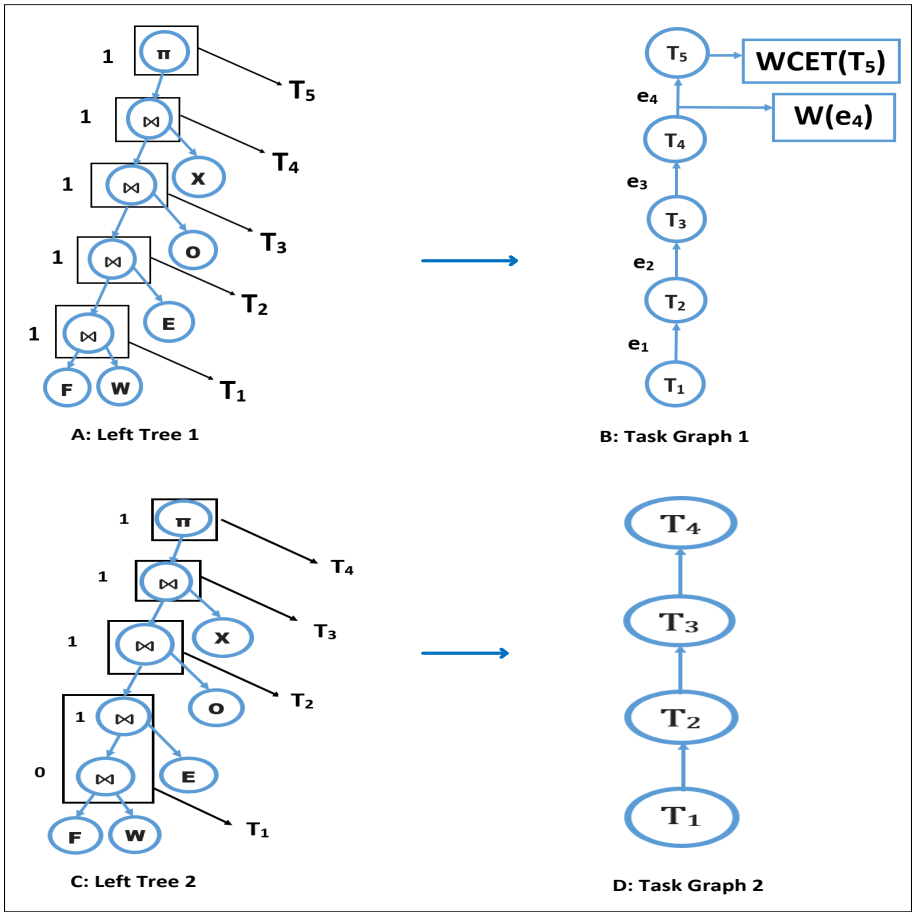


Figure 8.4: A:Left Deep Tree 1,B:Task Graph 1,C:Left Deep Tree 2,D: Task Graph 2

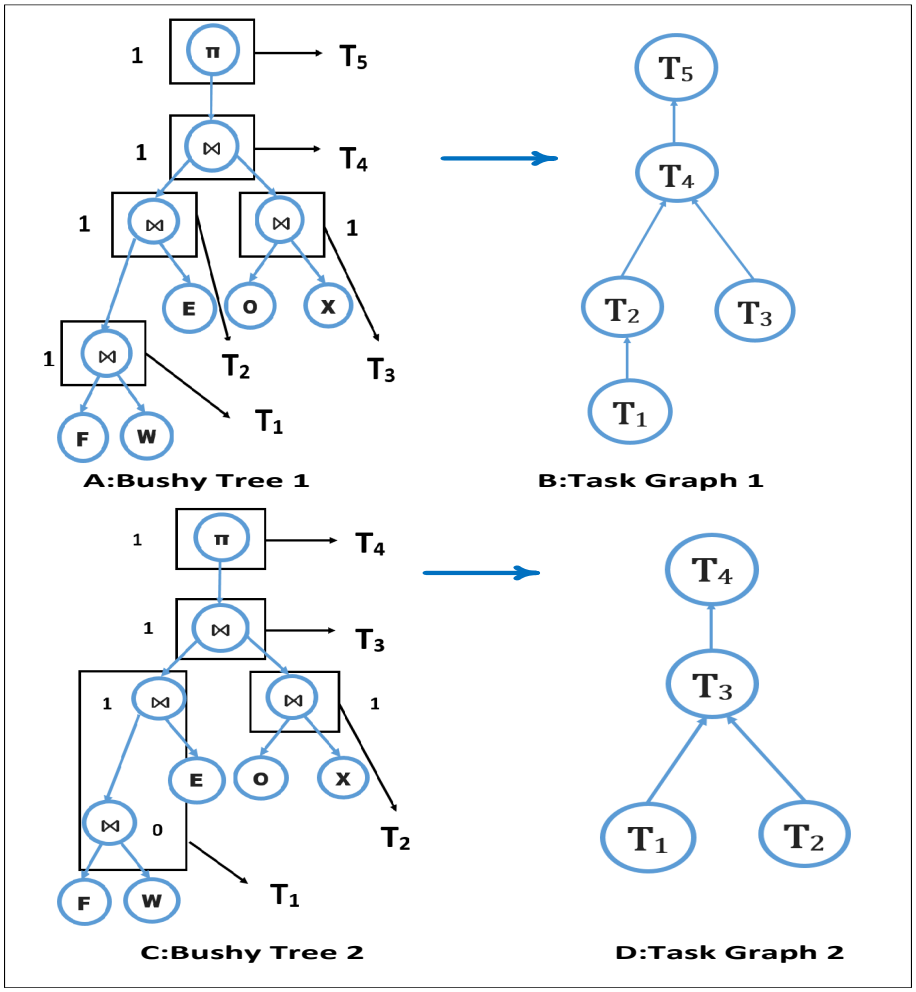


Figure 8.5: A:Bushy Tree 1, B:Task Graph 1, C:Bushy Tree 2, D: Task Graph 2

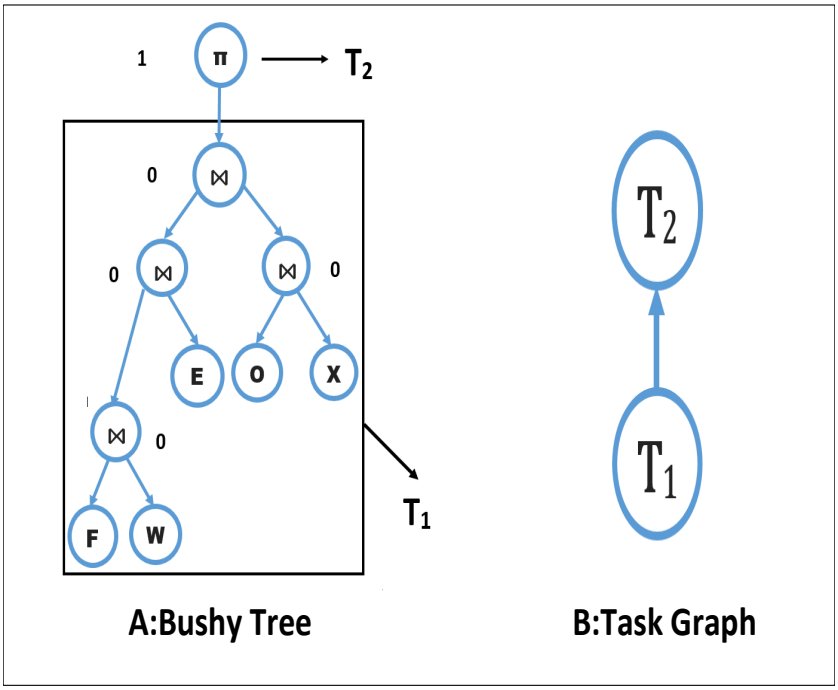


Figure 8.6: A:Bushy Tree , B:Task Graph

8.4.2 Query Tree

As described above we are considering two types of query trees including (i) LDT and (ii) BT. Fig.8.3 shows the informal representation of Q_D in the form of LDT and BT. Fig.8.2 shows the formal representation of different LDTs. Fig. 8.2A generates the LDT based TG having five nodes (T_1, T_2, T_3, T_4, T_5). Fig. 8.2B generates the LDT based TG having four nodes (T_1, T_2, T_3, T_4). Fig. 8.2C generates the LDT based task graph having three nodes (T_1, T_2, T_3). Fig. 8.2D generates the LDT based TG having two nodes (T_1, T_2). Fig. 8.2E generates the LDT based task graph having only one node (T_1).

8.4.3 Task Graph

The first task graph that is generated from the query tree considers each operation (i.e., join, projection) of a query tree as a one task (see Fig. 8.4B). For generating other task graph more than one operation of a query tree is considered as a one task (see Fig. 8.4D). As mentioned above that task graphs are generated depending on the type of query tree. They

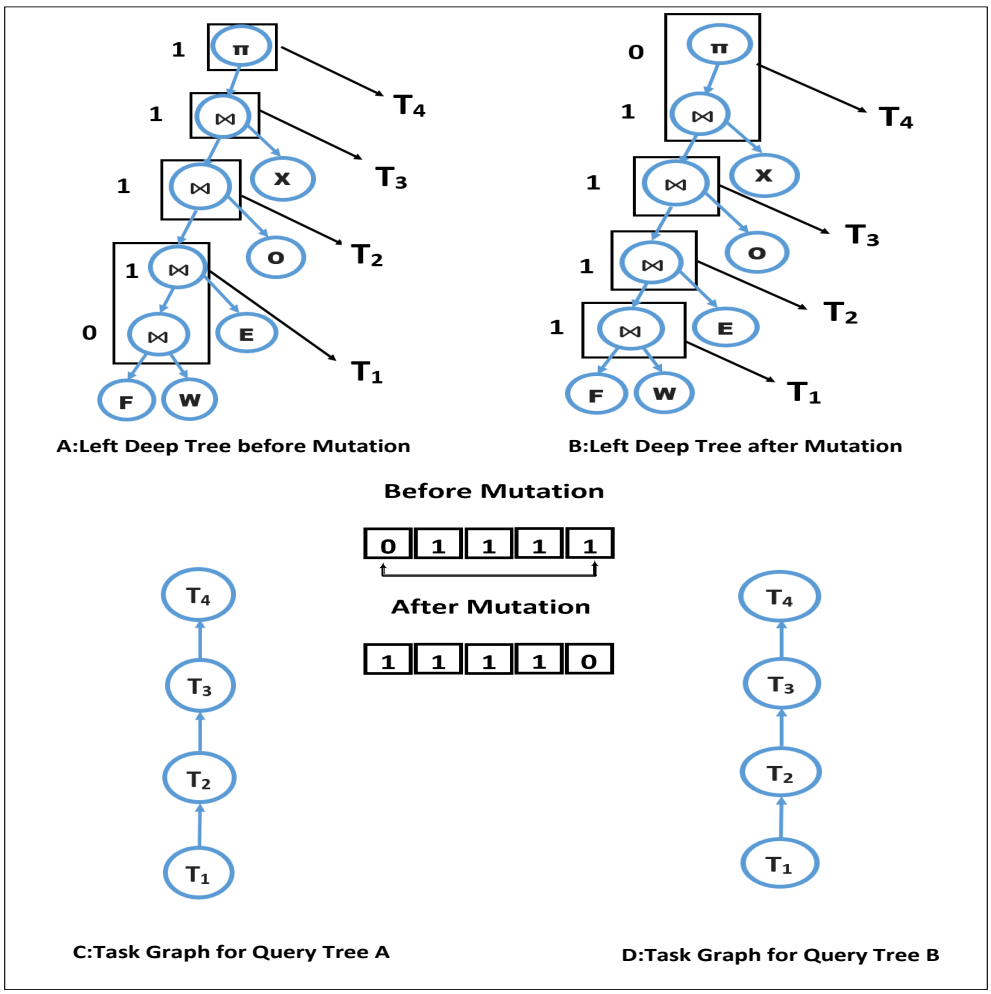


Figure 8.7: Left Query Tree before and after Mutation

can be LDT based TG or BT based task graph.

8.4.4 Left Deep Tree Based TG

If we consider Fig. 8.4A which contains the LDT created from Q_D, then it is seen that there is total of five operations (internal nodes) in this LDT.

Task Graph 1

The first and simpler task graph for the query tree in Fig. 8.4A is the task graph that contains the five nodes. The relational algebra equation for query Q_D is represented as follows.

$$\begin{aligned} &\{\Pi_{(oxygen.OxygenRatio, Water.WaterTemp} \\ &Oil.Oillimit, Engine.EngineID) \\ &((((F \bowtie W) \bowtie E) \bowtie O) \bowtie X))\} \end{aligned} \tag{8.1}$$

Now we will consider Eq. 8.1 for creating the task graphshown in Fig. 8.4B. In this TG each operation of the query tree is considered as a one separate task. This task graph comprises of five nodes. $T_1 = (F \bowtie W)$ $T_2 = \bowtie E$, $T_3 = \bowtie O$, $T_4 = \bowtie X$, $T_5 = \Pi_{(oxygen.OxygenRatio, Water.WaterTemp, Oil.Oillimit, Engine.EngineID)}$.

Task Graph 2

Now we will generate another task graph from the query tree shown in Fig. 8.2C. We consider the Eq. 8.1 for creating this task graph but the operations $(F \bowtie W)$ and $\bowtie E$ are combined into one task T_1. So the nodes defined in Fig. 8.4D become $T_1 = (F \bowtie W) \bowtie E$, $T_2 = \bowtie O$, $T_3 = \bowtie X$, $T_4 = \Pi_{(oxygen.OxygenRatio, Water.WaterTemp, Oil.Oillimit, Engine.EngineID)}$.

If we compare the LDT in Fig. 8.4A with LDT in Fig. 8.4C, we can see that the LDT based task graph in Fig. 8.4D contains the two join operations in task node T_1. In contrast, the LDT based TG in Fig. 8.4B only contains the one join operation in task node T_1. Therefore the the task graph in Fig. 8.4B has five nodes and TG in Fig. 8.4D has four

nodes. The total number of these TGs keeps on increasing as long as the number of joins and relations in FDQ increases. Similarly, other TGs can be created.

8.4.5 Bushy Tree Based TG

In this section we elaborate the creation of BT based TGs.

Task Graph 1

Fig. 8.5A represents the BT. This BT is represented by Eq. 8.2 which is the relational algebra representation of Q_D.

$$\begin{aligned} &\{\Pi_{(oxygen.OxygenRatio, Water.WaterTemp} \\ &Oil.Oillimit, Engine.EngineID) \\ &[((F \bowtie W) \bowtie E) \bowtie (O \bowtie X)]\} \end{aligned} \tag{8.2}$$

Eq. 8.2 is helpful in creating the task graphs. The algebraic representation of the BT based task graph (see Fig. 8.5A) is consider as $T_1 = (F \bowtie W)$, $T_2 = \bowtie E$, $T_3 = \bowtie (O \bowtie X)$, $T_4 = (((F \bowtie W) \bowtie E) \bowtie (O \bowtie X))$, $T_5 = \Pi_{(oxygen.OxygenRatio, Water.WaterTempOil.Oillimit, Engine.EngineID)}$. This TG is shown in Fig. 8.5B.

Task Graph 2

Another query tree generated for the BT is shown in Fig. 8.5C. The task graph that is created from the Fig. 8.5C is shown in Fig. 8.5D. Therefore Fig. 8.5D comprises a lower number of nodes as compared to the task graph shown in Fig. 8.5B. These tasks are $T_1 = ((F \bowtie W) \bowtie E)$, $T_2 = \bowtie (O \bowtie X)$, $T_3 = (((F \bowtie W) \bowtie E) \bowtie (O \bowtie X))$, $T_4 = \Pi_{(oxygen.OxygenRatio, Water.WaterTempOil.Oillimit, Engine.EngineID)}$.

Task Graph 3

Similarly, another task graph is also created. This task graph contains tasks including $T_1 = (F \bowtie W) \bowtie E) \bowtie (O \bowtie X)$, $T_12 = \Pi_{(oxygen.OxygenRatio,Water.WaterTempOil.Oillimit,Engine.EngineID)}$. So this task graph only contains two nodes, as shown in Fig. 8.6B. While the Bushy tree for this task graph is shown in Fig. 8.6A. So there is a large number of task graphs that can be created using a similar technique. All these TGs are part of the solution space of the GA. The size of the solution space increases depending upon the number of task graphs.

8.4.6 Implementation of Genetic Algorithm for Our Example

This section elaborates the steps included in the implementation of GA in our context.

Initial Population

Each task graph (individual solution) is created by considering the different combinations of query tree operations. As described above, two types of task graphs are generated depending on the type of query tree. These task graph are LDT based TG and BT based task graph. The chromosome representation of these task graph are shown in Fig. 8.4. The task graph based on LDT are shown in Fig. 8.2. and the task graph based on the BT is shown in Fig. 8.5 and Fig. 8.6.

Fitness Function

The WCET of each task node within the TG along with the weight of edges is given as an input to the scheduler for the calculation of the fitness function i.e., (makespan). For illustration, we present an example with three task graphs (see Fig. 8.4B, Fig. 8.4D, Fig.8.5B). The input given to the scheduler for calculating the fitness function (makespan) of each TG is shown in Table. 8.1.

Table 8.1: Makespan of TG (Secs)

Task Graph	WCET (ms)	W(e) MB
Fig. 8.4B	$T_1 = 1.08$ms	58.80
	$T_2 = 3.12$ms	$T_1 \rightarrow T_2 = 33.78$MB
	$T_3 = 4.56$ms	$T_2 \rightarrow T_3 = 67.24$MB
	$T_4 = 5.08$ ms	$T_3 \rightarrow T_4 = 45.98$MB
	$T_5 = 4.23$ms	$T_4 \rightarrow T_5 = 78.90$MB
Make Span	7.29ms	
Fig. 8.4D	$T_1 = 4.35$	
	$T_2 = 4.56$ms	$T_1 \rightarrow T_2 = 67.24$MB
	$T_3 = 5.08$ms	$T_2 \rightarrow T_3 = 45.98$MB
	$T_4 = 4.23$ms	$T_3 \rightarrow T_4 = 78.90$MB
Make Span	6.09ms	
Fig.8.5B	$T_1 = 1.08$	
	$T_2 = 4.87$ms	$T_1 \rightarrow T_2 = 58.80$MB
	$T_3 = 5.48$ms	$T_2 \rightarrow T_4 = 56.03$MB
	$T_4 = 4.23$ms	$T_3 \rightarrow T_4 = 66.19$MB
	$T_5 = 4.74$ ms	$T_4 \rightarrow T_5 = 78.90$MB
Make Span	4.18ms	

Selection

Our selection phase selects the fittest individuals and passes them to the next generation. For each solution, makespan is calculated. The solutions with minimum makespan are selected. According to Table. 8.1, the solution in Fig. 8.5B has the smallest makespan. The makepsan of solutions are compared to each other. The solutions with minimum makespans are considered as the fittest solutions and are passed to the next generation.

Mutation

In the case of the proposed solution, the mutation is only performed for the fittest solutions. If two task graphs are mutated and the resultant child task graph is similar to the the one of its parent then that task graph is not considered for the next generation. Fig. 8.7A shows the LDT before mutation and Fig. 8.7B shows the LDT after the mutation. Therefore Fig.

8.7C shows the TG for the LDT in Fig. 8.7A and Fig. 8.7D shows the TG for the LDT in Fig. 8.7B. It is clear that both task graphs showed in Fig. 8.7C and Fig. 8.7D are similar in structure, but it is clear that the T_1 in Fig. 8.7C is different from the T_1 in Fig. 8.7D. It means that the overall WCET of T_1 in Fig. 8.7C is different from the WCET of T_1 in Fig. 8.7D, which changes the makespan of both TGs.

8.5 Calculation of WCET using Example Query

This section shows the calculation of the WCET of the task graph shown in Fig. 8.4B. For example the FDQ Q_D has four sub-queries and they are named as Q_{D1}, Q_{D2}, Q_{D3}, Q_{D4}. Q_{D1} represents the task node T_1. Q_{D2} represents the task node T_2. Q_{D3} represents the task node T_3. Q_{D4} represents the task node T_4. In this case only join queries are considered.

Q_{D1}: Select Oxygen.OxygenRatio,Water.WaterTemp,Oil.Oillimit, Engine.EngineID from Oxygen, Oil, Water,Engine,Fuel where Fuel.WaterID=Water.WaterID and Water.WaterID =Engine.EngineID and Oil.OilID=Engine.OilID and Oxygen.OxygenID=Oil.OxygenID

Q_{D2}: select Oxygen.OxygenRatio, Water.WaterTemp,Oil.Oillimit, Engine.EngineID from Oxygen, Oil, Water,Engine, Fuel where Fuel.WaterID=Water.WaterID and Water.WaterID =Engine.EngineID and Oil.OilID=Engine.OilID

Q_{D3}: Select Oxygen.OxygenRatio, Water.WaterTemp,Oil.Oillimit, Engine.EngineID from Oxygen, Oil, Water,Engine,Fuel where Fuel.WaterID=Water.WaterID and Water.WaterID =Engine.EngineID

Q_{D4}: Select Oxygen.OxygenRatio, Water.WaterTemp, Oil.Oillimit, Engine.EngineID from Oxygen, Oil, Water,Engine,Fuel where Fuel.WaterID=Water.WaterID

According to Fig. 8.3B and Eq. 8.2 $Q_{D1} = T_5$, $Q_{D2} = T_4$, $Q_{D3} = T_3$, $Q_{D4} = T_2$. For each of these FDQs the estimated WCET is determined by running these FDQs with different sets of data. Almost 3,000 FDQs are run and their WCET is calculated. The WCET calculated for the task graph in Fig. 8.4B is shown in Table. 8.1.

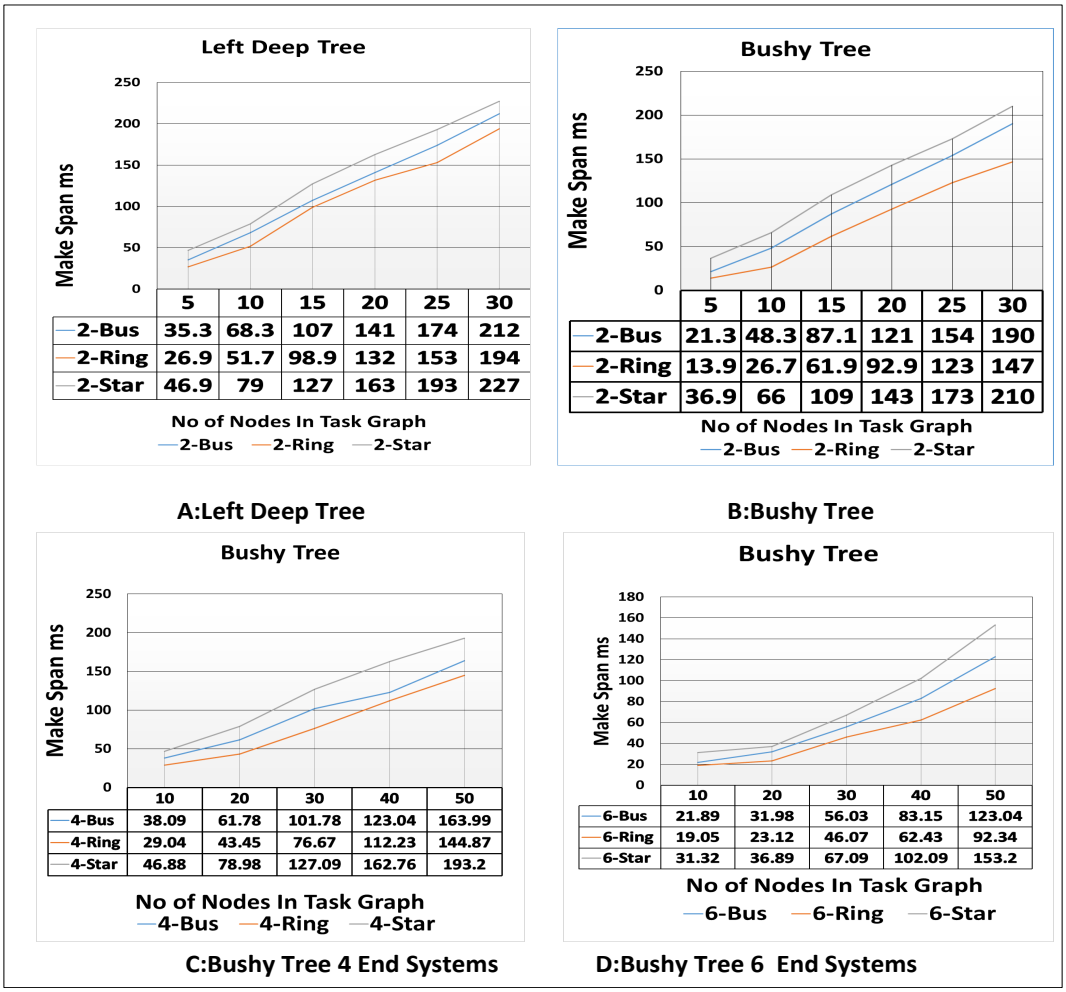


Figure 8.8: A:LDT based TG, B:BT based TG, C:BT based TG with Four processors, D: BT based TG with Six Processors

8.6 Results

In experimental results, three different types of network topologies including (i) star-bus (ii) star-ring and (iii) star-star are considered. The proposed GA selects the TG with minimum makespan in all the generations. All the selected task graphs are given as an input to the scheduler for calculating the makespan. The scheduler is running over the networked distributed systems based on the three network topologies.

Fig. 8.8A and Fig. 8.8B shows the result in the case where two end systems in the network topology are considered. If both results are compared, it is clear that the TGs

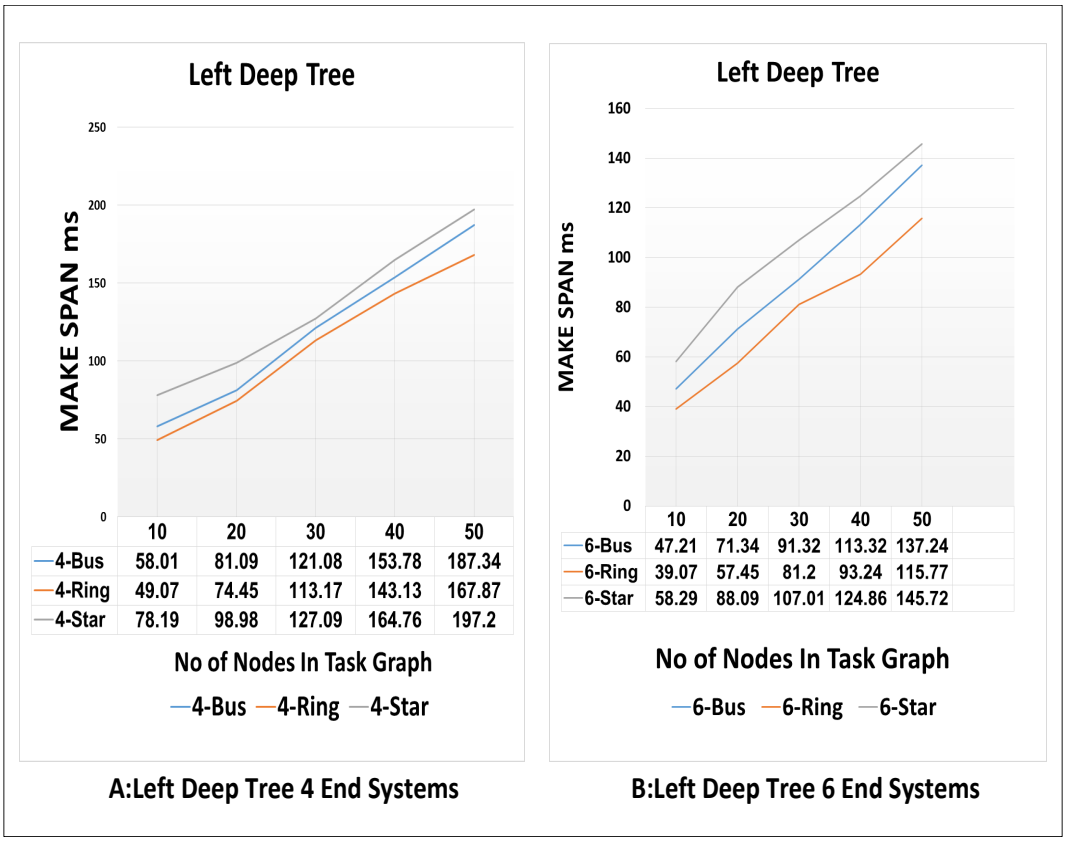


Figure 8.9: A:LDT Based TG with Four End Systems, B:LDT Based TG with Six End Systems

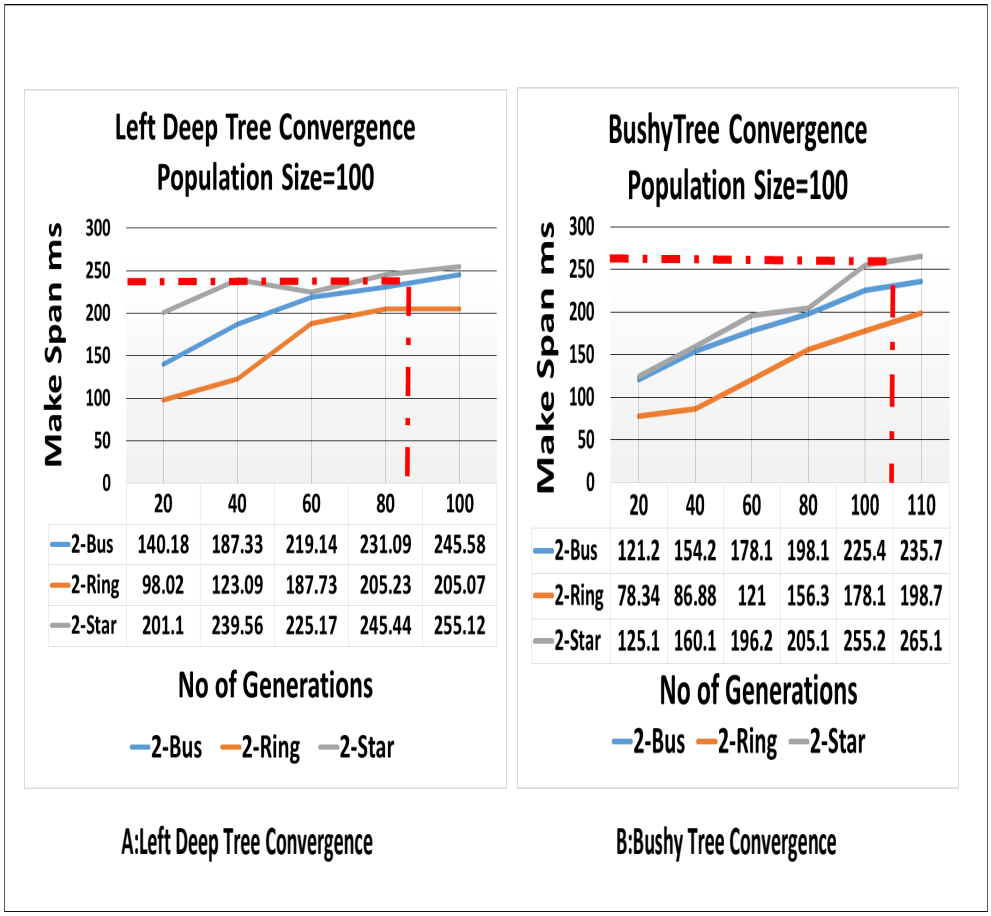


Figure 8.10: A:LDT Convergence, B:BT Convergence

generated on the basis of the BT have better makespans as compared to the LDTs. In case of LDTs there is a linear graph because the scheduler can process only one task node at one time because each child task has to start its execution after its parent task completed its execution, so the scheduler cannot schedule multiple tasks. On the other hand in case of the BT there are more tasks at one level of the TG which are ready for processing so at one point of time multiple tasks can be assigned to processing nodes due to which the overall makespan is reduced.

If the results presented in Fig. 8.8C and Fig. 8.8D are compared then it is clear that the BT gives better results since the makespan of the task graphs with more than 30 nodes shows a remarkable decrease in the case when there are more end systems. The task graphs with a lower of nodes do not show any reduction in makespans when more end systems are added. If Fig. 8.9A and Fig. 8.9B are compared then it is clearly seen that in case of LDTs the makespan is also decreased when more end systems are added in the topology. So the TGs based on LDT have better makespan in case the scheduler uses six end systems.

If Fig. 8.8C and Fig. 8.9A are compared, then it is clearly seen that the BT based task graphs have better results in case of 4 end systems as compared to the LDT based TG. Similarly, if the Fig. 8.8D and Fig. 8.9B are compared, it is also clear that the BT based task graphs has a lower makespan as compared to the LDT based task graphs in case the scheduler uses six end systems. If Fig. 8.10A and Fig. 8.10B are compared it is evident that the BT based search space converges earlier as compared to the LDT based search space. In the case of LDT, the GA converges after 80th generations while in case of the BT it converges after 100 generations but gives a better solution with lower makespans.

If we consider the different results based on different topologies then it is evident that ring topology leads to reduced makespan as compared to star and bus topologies. In case of the ring topology there are more connections between the end systems so the overall network load is minimized. Minimized network load increases the chances of scheduleability.

CHAPTER 9
FAULT DETECTION AND DIAGNOSIS USING DMG FOR SAFETY CRITICAL HVAC SYSTEMS

This chapter evaluates the previously introduced algorithms for query optimization in a real-world use case with an Heating Ventilation and Air Conditioning System (HVAC). The HVAC system depends on diagnostic query-based fault detection and diagnosis to monitor faults. It comprises both safety-critical services for emergency situations (e.g., building fire) and non safety-critical services for improved comfort and efficiency.

9.1 Background of HVAC Systems

In the U.S., heat, ventilation and cooling (HVAC) systems account for roughly 43% of the overall energy consumption in buildings [130]. Researchers have tried to improve these numbers by deploying more embedded sensors in the systems to monitor temperature, CO_2 and humidity levels [131] but including electrical components has made the systems failure-prone. If a fault arises in one of the electrical components of such systems, the sensors may produce erroneous data and the actuators may behave differently than what is expected. These failures usually lead to general human discomfort, excessive energy consumption, increased overall operation costs, and deterioration of equipment lifespan. Regular checks and maintenance solve this problem but because of the increased cost of on-site maintenance, preventive or predictive maintenance in the form of fault-tolerant systems has become much more significant in recent years [132, 133, 134, 135, 136, 137].

Along with controlling the indoor environment levels, the HVAC systems are generally integrated with hazard detectors such as smoke detectors and play a crucial role in restricting hazardous situations. In hospitals, for example, the HVAC systems are one of the primary responders in the case of a fire [138]. In such situations, it is highly essential

that the HVAC systems can discern a critical from a non-critical situation i.e. they should identify if an actual fire has occurred or if there is some fault within the system. False alarms have a much more negative impact than expected. For example, in homes, occupants have to search for the detector that is sounding the alarm and determine themselves if the alarm is false or an actual hazardous situation exists. This process is time-consuming, stressful and potentially dangerous. Ordinarily, the alarms may mistake normal situations for critical ones. For example, the fire alarms near the bathrooms mistake the steam from the shower as smoke. In these situations, inhabitants may shut the detectors off to avoid the annoyance of false alarms. The malfunctioning of the detection system itself is equally disastrous in life-threatening situations. For example, faulty readings from CO_2 sensors in HVAC systems lead to undetected air poisoning that kills approximately thousands of people each year. Therefore, disabled or ineffective detectors may cause a high death toll and property damage from hazardous situations that otherwise could have easily been prevented [139].

9.1.1 Time Sensitivity of HVAC Systems

In hazardous situations, thousands of lives depend upon the correct functioning and timely response of the HVAC systems. If a component is faulty, it should be detected before a critical situation occurs and actions, such as ventilation and closing of doors/windows/etc depending on the locations of fires, smoke, people should be performed within in the shortest time period provided by the system. It means that the HVAC systems are time-sensitive and the fault detection and diagnosis technique (FDD) used should identify faults within the system provided time-frame. This aspect of the HVAC systems, although essential, is usually not discussed in the literature.

9.1.2 Contribution

In this chapter, we exploit the query optimization techniques proposed in the previous chapters for a time-critical fault detection and diagnosis approach for HVAC systems. We consider an HVAC system integrated into a complex building architecture with multiple rooms and floors. There are different sensors and actuators in each room. The HVAC system maintains the indoor environment levels and also provides detection for fire hazards. We detect faults in the sensors using diagnostic queries that measure the values of the sensors for critical and non-critical situations. We consider the critical situations time-sensitive and our FDD technique determines whether the fire has occurred or a sensor has malfunctioned within the system's defined time-bound. The detection and diagnosis in non-critical situations has no time restriction and is done to minimize the energy consumption of the HVAC systems due to faulty equipment. To the best of our knowledge, time-sensitive FDD has never been discussed in HVAC systems before and is the main contribution of our work. Moreover, our technique does not depend upon the architecture of the building and is integrable with any HVAC system comprising sensors and actuators.

9.2 Problem Formulation

In this work, we propose a multi-query based fault detection and diagnosis (FDD) technique for HVAC systems. We consider a complex building architecture with multiple rooms and floors. Each room has a certain set of sensors and actuators depending upon its requirements and function. We consider fault detection and diagnosis in the HVAC systems in two separate scenarios: i). in rooms that have fire hazardous equipment, e.g. kitchen and ii). in rooms where there is no possibility of a fire, e.g. study room. The functioning of the HVAC system in a fire-prone environment is considered a critical situation and is highly time-sensitive. It means that the faults in the electrical equipment in such situations should be identified and rectified within the system's defined time-bound. In contrast, the

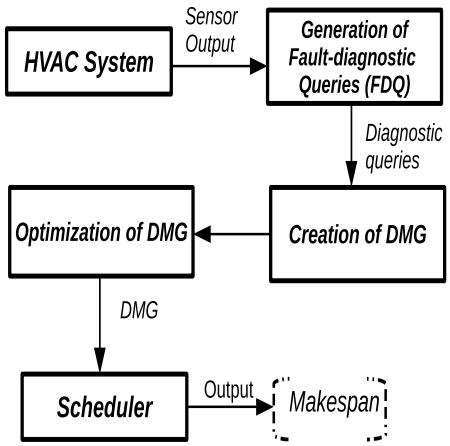


Figure 9.1: Flow diagram of diagnostic query based fault detection and diagnosis

maintenance of comfort levels (e.g. room temperature) by the HVAC systems is considered non-critical and has no time restriction for fault detection and diagnosis. The sensors and actuators in each room provide output data that is stored in a central database. We formulate diagnostic queries that compare the sensor and actuator values with certain thresholds and concentration levels to determine faults in the system. Since we have multiple sensors and actuators and thus multiple sets of queries, we formulate a diagnostic multi-query graph (DMG) for a structured and timely diagnosis. Considering we are dealing with highly time-critical situations where a false output could cause a dangerous situation, it is essential that we know beforehand the maximum time that our FDD may take to identify faults i.e. the total execution time of the DMG. For this purpose, we give this DMG to an off-line scheduler that allocates, maps, and schedules the queries and gives an execution time for fault detection. This execution time is termed makespan. This makespan should always be less than the deadline defined by the system for the detection of fire and people in the building. To ensure that we meet this deadline, we optimize the DMG before giving it to the scheduler to get an optimal schedule for the execution of the DMG. Fig. 9.1 gives the main steps of our approach.

9.2.1 Building Architecture

We consider a building with multiple rooms, corridors, and floors. There are two main components for modelling the building, (i) the HVAC model and (ii) FDQ execution model.

HVAC model

Each room and corridor in the building has a certain set of sensors and actuators depending upon its type, function, and requirements for fault detection. We divided the rooms into two categories,

- **Catastrophic Rooms (CR):** All the rooms that contain fire hazardous equipment are sorted into this category. In such rooms, the HVAC system detects the possibility of a fire and sounds the alarm if a fire has occurred. The fault detection and diagnosis in such rooms is highly time critical because a faulty component can either hide the existence of an actual fire or can sound a false alarm that creates a stressful and dangerous situation. Therefore, in such rooms, it is important that we identify a defective component within the system's defined deadline before a catastrophic situation occurs. We consider the following sensors and actuators in such rooms.

  - *Temperature:* A sensor that measures the temperature of the room and the adjacent corridor.
  - CO_2*:* A sensor that measures the concentration level of carbon dioxide in the room.
  - *Heater (thermostat):* An actuator that controls the amount of heat in the room.

  We consider that a fault can occur in any of the mentioned components. If the output from all three of the components point to the occurrence of a fire then the system sounds an alarm whereas if one of the components is giving a value above its threshold or is inconsistent with the values of the other two components then the system

classifies it as defective. Here, the component needs fixing or replacement. To simplify the situation, we assume that only one of the electrical components may be defective at a time.

- **Normal Rooms (NR):** The unavailability of fire hazards classifies a room into this category. These rooms require the normal functioning of the HVAC systems, i.e. controlling temperature and humidity levels to provide a comfortable environment for a living. The fault detection in such cases is not time-critical and thus is not restricted by any deadline. However, it is still essential to identify the faults since a defective component adds to the energy consumption of the HVAC system and causes general discomfort for the inhabitants. NR have the same set of sensors as in CR but we use the damper instead of heater. It deals with the air flow calculations of the ventilation system and takes the input from the CO_2 sensor. Similar to CR, we assume that only one component is faulty at a time.

For example, we would classify the chemistry lab in the HVAC model of a school building as a catastrophic room (CR) because it has various sensitive chemicals that can cause explosives whereas we would classify the gym into a normal room (NR) since it has no fire-prone equipment. We have used the method described in [140] to compose our HVAC model in MATLAB/Simulink. For simplification of the model, we have restricted the number of CR per floor to one. The rest of the rooms and the corridor (C) on the floor are classified as NRs. There is no restriction on the number of rooms per floor or the number of floors in the building and our technique is scalable to any kind of building structure provided the rooms can be classified into one of the mentioned categories. Fig. 9.2 shows the HVAC model for a two floor building each with three rooms and one corridor.

9.2.2 Formulation of DMG

With multiple rooms in the building each with its own set of sensors and actuators, it is difficult to keep track of the data and the execution of the corresponding diagnostic

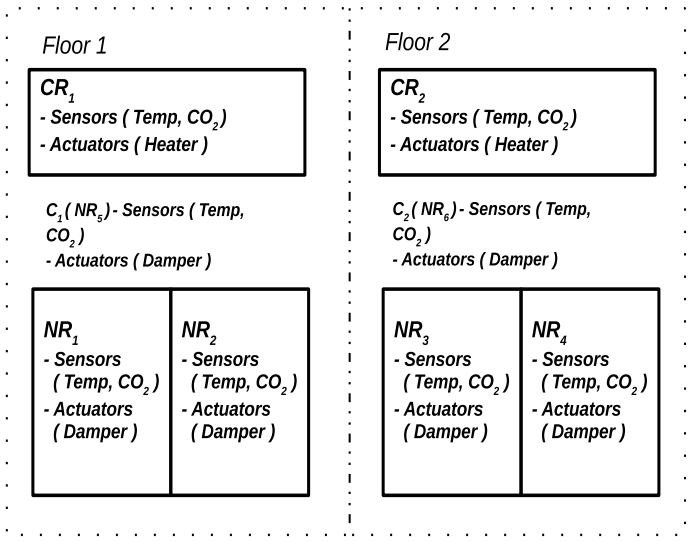


Figure 9.2: HVAC model of a two floor building each with three rooms and one corridor

queries. To simplify this process, we formulate a diagnostic multi-query graph (DMG) that represents all the queries and their different relationships in a structured form that is much easier for the scheduler to schedule and for the processing elements to execute. A basic DMG in our context is shown in Fig. 4.2.

It is important to mention how different DMGs with multiple FDQs are created based on sensors present in each room. For each room (NR or CR) a separate DMG is created. This DMG is based on the FDQ about the fault which we want to detect in a particular room. Therefore, for each room one DMG is created. Later on, all these separate DMGs (one for each room) are combined and one DMG for a particular floor is formulated. We have considered only two floors for the simplification of our problem. The next section will cover the important steps involved in the formulation of these DMGs.

Step 1A: Creation of FDQs for Floor 1 (F_1)

The FDQs written for NR_1 and NR_2 present at F_1 are shown in Table 9.1 and Table 9.2. In the case of normal rooms, we will detect two types of faults (i). sensor faults (temperature and CO_2 sensors), (ii). concentration of CO_2 and temperature in a room. In the case of normal rooms, the concentration of CO_2 and temperature is only checked for maintaining

the comfort level of the rooms. This type of fault detection is not time-sensitive. So in the case of normal rooms, we are not detecting the critical faults such as fire. And if these faults are not detected in stringent timing deadlines provided by the system then no catastrophic situation will occur. Therefore, fault detection performed in normal rooms will minimize the overall consumption of energy and resources by implementing timely fault detection and diagnosis mechanism. It is also important to mention that we are only measuring one fault at one instant of time. The FDQs in normal rooms are executed after every one hour as we are not dealing with highly critical faults.

FDQs for finding CO_2 Sensor Faults $NR_1(F_1)$

If we analyze the FDQs mentioned in Table 9.1 it is seen that the value of $CO_2faultcounter = 1$ which shows that there is a fault in the CO_2 sensor. The other parameters including (i) room_temperature1 and (ii) $CO_2concentration$ have also values greater than their threshold (Threshold values: $CO_2 > 399ppm$ and $temperature > 19C$) which shows that there is a possibility that these sensors are also faulty. But from FDQ RQ_{14} it is evident that the CO_2 sensor present in NR_1 is faulty at the moment.

FDQs for finding CO_2 and Temperature Concentration $NR_2(F_1)$

Table 9.2 shows the FDQs written for finding the concentration of CO_2 and temperature in NR_2. If we analyse the FDQ (RQ_{21}) then we can see that the value of $CO_2faultcounter$ is zero. And the values of room_temperature2 and $CO_2concentration$ are greater than their thresholds (RQ_{22}, RQ_{23}). Therefore, FDQs including RQ_{22} and RQ_{23} shows that there is higher values of CO_2 and temperature in the room NR_2 so the window of the room should be open in order to maintain the comfort level of the room.

Table 9.1: FDQ_{11} for CO_2 Sensor fault (NR_1) F_1

Names	FDQS
RQ_{11}	Select $CO_2faultcounter$ from $CO_2sensor$ where $CO_2faultcounter = 1$
RQ_{12}	Select CO_2value , $CO_2faultcounter$ from $CO_2sensor$ where $CO_2concentration > 399$ and $CO_2faultcounter = 1$
RQ_{13}	Select room_temperature1 $CO_2faultcounter$ from rooms, $CO_2sensor$ where rooms.id= $CO_2sensor.id$ and room_temperature1>19 and $CO_2faultcounter = 1$
RQ_{14}	Select $CO_2faultcounter$, CO_2value room_temperature1 from $CO_2sensor$, rooms, where $rooms.id = CO_2sensor.id$ and room_temperature1>19 and $CO_2concentration > 399$ and $CO_2faultcounter = 1$

Step 1B: Creation of FDQs for Floor 2 (F_2)

In this section, the FDQs are created for finding the fault in the temperature sensor in a room NR_3 present on floor 2. Table 9.3 shows the FDQs (RQ_{31}) in which the fault in temperature sensor is detected. The value of temperaturefaultcounter is one in RQ_{31} which shows that there is a fault in the temperature sensor rather than the fact that $CO_2concentration$ and temperature have values which are greater than their thresholds. Therefore, in Table 9.4 the FDQs are created for finding the concentration level of CO_2 and the value of temperature in the room NR_4. The value of temperaturefaultcounter is zero which shows that there is no fault in the temperature sensor. However, the values of $CO_2concentration$ and temperature are greater than their threshold. This means that the inhabitant should open the window to maintain the comfort level of NR_4.

Step 2: Creation of DMG from FDQs

In this step the DMG is created from the FDQs generated in step 1. DMG_{11} for NR_1 is created from the FDQ_{11} presented in Table 9.1. DMG_{12} for NR_2 is created from the FDQ_{12} presented in Table 9.2. Both DMG_{11} and DMG_{12} are joined together (named as (DMG_{F1})

Table 9.2: FDQ_{12} for CO_2 and Temperature Levels in NR_2 F_1

Names	FDQS
RQ_{21}	Select $CO_2faultcounter$ from $CO_2sensor$ where $CO_2faultcounter = 0$
RQ_{22}	Select CO_2value , $CO_2faultcounter$ from $CO_2sensor$ where $CO_2concentration > 399$ and $CO_2faultcounter = 1$
RQ_{23}	Select room_temperature2 $CO_2faultcounter$ from rooms, $CO_2sensor$ where rooms.id= $CO_2sensor.id$ and room_temperature2>19 and $CO_2faultcounter = 1$
RQ_{24}	Select $CO_2faultcounter$, $CO_2concentration$ room_temperature2 from $CO_2sensor$, rooms, where $rooms.id = CO_2sensor.id$ and room_temperature2>19 and $CO_2concentration > 399$ and $CO_2faultvaluecounter = 1$

for finding the two faults (i). CO_2 sensor fault, (ii) CO_2 and temperature levels in rooms at F_1. Similarly DMG_{13} for NR_3 is created from the FDQ_{13} presented in Table 9.3. DMG_{14} for NR_4 is created from the FDQ_{14} presented in Table 9.4. Both DMG_{13} and DMG_{14} are joined together (named as DMG_{F2}) for finding the faults including (i) temperature sensor fault and (ii) CO_2 and temperature values in rooms at F_2. Fig 9.3 shows the process of creating the DMG based on FDQs for each floor. We have created different DMGs for each floor because we want to compare the temperature of one floor to another floor. The resultant tuples (RT) generated by the DMG for different floors are compared. If the RT of one floor is different from the RT of another floor then there is a possibility that one of the floors has a fault either in its sensor or has higher values of temperature or CO_2.

Step 3: Execution of DMG

In this step the DMG (comprised of the FDQs) is executed by the processors according to the plan given by the scheduler [122]. As we have created two DMGs in Step 2 named as (i). DMG_{F1} and (ii) DMG_{F2}. Both of these DMGs have different FDQs. After these FDQs are executed over the database (by processors) the final FDQs from each DMG (RQ_{14},

Table 9.3: FDQ_{13} for Temperature Sensor fault for NR_3 F_2

Names	FDQS
RQ_{31}	Select Temperaturefaultcounter from Temperaturesensor where Temperaturefaultcounter='1'
RQ_{32}	Select $CO_2concentration$, Temperaturefaultcounter from $CO_2sensor$ C, Temperaturesensor T where C.id=T.id $CO2concentration > 399$ and $Temperaturefaultcounter = 1$
RQ_{33}	Select room_temperature3 Temperaturefaultcounter from rooms, $CO_2sensor$ where rooms.id= $CO_2sensor.id$ and room_temperature3>19 and Temperaturefaultcounter='1'
RQ_{34}	Select Temperaturefaultcounter, $CO_2concentration$ room_temperature3 from $CO_2sensor$, rooms, where $rooms.id = CO_2sensor.id$ and room_temperature3>19 and $CO_2concentration > 399$ and Temperaturefaultcounter='1'

RQ_{24}, RQ_{34}, RQ_{44}) generates some resultant tuples (i.e., fault values) for both floors.

Step 4: Fault Detection

The important parameters which are considered for fault detection are shown in Table 9.5. These parameters are defined as follows: (i) Resultant Tuples (RT) are the results (tuples) extracted from the execution of FDQs on each floor. The $RT = 1$ means that there are some faulty values in RT from both floors. (ii) RT extracted from FDQ of one floor is compare with the RT on the other floor. This term is named as comparison of RT between two floors (CA). If there is a similarity between the RTs of two floors then it is clear that both floors have some fault in CO_2 sensor. The numeric value $CA = 1$ shows that both floors have similar faulty values while $CA = 0$ shows that there is no similarity between the RTs of both floors. (iii) Threshold values (TV) is used to match the RT with the TV stored in the database. For example for the FDQ_1 the threshold value for $CO_2Sensorvalue > 399$. If there extracted $RT > TV$ then we will assign 1 (true to this parameter) which shows that

Table 9.4: FDQ_{14} for CO_2 and Temperature Levels in NR_4 F_2

Names	FDQS
RQ_{41}	Select Temperaturefaultcounter from Temperaturesensor where Temperaturefaultcounter='0'
RQ_{42}	Select $CO_2concentration$, Temperaturefaultcounter from $CO_2sensor$ C, Temperaturesensor T where C.id=T.id $CO_2concentration > 399$ and Temperaturefaultcounter='0'
RQ_{43}	Select room_temperature3 Temperaturefaultcounter from rooms, $CO_2sensor$ where rooms.id= $CO_2sensor.id$ and room_temperature3>19 and Temperaturefaultcounter='0'
RQ_{44}	Select Temperaturefaultcounter, $CO_2concentration$ room_temperature3 from CO2sensor, rooms, where $rooms.id = CO_2sensor.id$ and room_temperature3>19 and $CO_2concentration$¿399 and Temperaturefaultcounter='0'

Table 9.5: Fault Parameters

	RT	*CA*	*TV*	Fault (Yes/No)
DMG_{F1}	1	1	1	Yes
DMG_{F2}	1	1	1	Yes
DMG_{F1}	1	0	0	No
DMG_{F2}	0	1	0	No

there is a fault in the sensor. In this manner, we can detect the fault in the CO_2 sensor.

Hence in a similar manner, all the DMGs are created and different types of faults including different sensors can be detected.

9.2.3 Optimization Technique

In the previous section, we have described how different DMGs are created based on FDQs generated for each floor. Increasing the number of rooms in the building will also increase the number of sensors. An increasing number of sensors means that we have a large number of inputs that need to be processed for finding faults in the HVAC system. Hence we have to write thousands of FDQs for considering the sensor faults from the complete building.

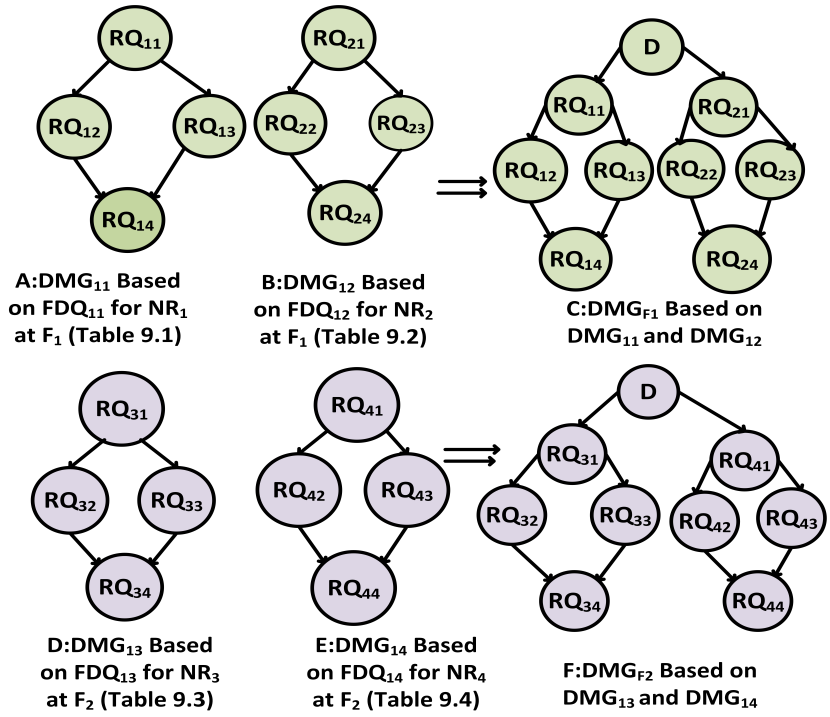


Figure 9.3: DMG Formation of Normal Room (NR)

Due to this reason, the overall process of fault detection and diagnosis (processing of FDQ queries in each DMG from the different rooms) becomes extremely complex and costly in terms of time, as we have to process the inputs from higher numbers of sensors present at each floor (inside each room). It is also possible that some of these FDQs are time-sensitive and have stringent timing requirements. This is the scenario when a system has to deal with hard real-time requirements. One example of this scenario is the fire in the building and the HVAC system has to shut down as soon as possible. In this scenario, all the FDQs should complete their execution within the stringent deadline provided by the system. Therefore to meet the stringent timing requirements we are applying a GA based optimization to our DMGs so that they can complete their execution before any catastrophic situation may occur. The proposed method of optimization based on the GA is described in Algorithm 5.

Algorithm 5 DMG Optimization Algorithm

Input: DMG with DFQs
Output:Optimized DMG with minimum Makespan

Joins methods (JM)=Cross join, Hash Join, Loop Join, Merge Join
for all Nodes in DMG **do**
    for each $q_i \in Node(DMG)$ **do**
        for each join $j_k \in JM$ **do**
            Apply j_k to q_i
            Apply Best Join order (JO) to q_i
            Generate a new DMG (Solution S_i)
            Add S_1 to Search Space (SP) of GA
        end for
    end for
end for
Create initial population based on these $S_i \in S$ for GA.
while the S_i with minimum Makespan is not found **do**
    for all S_i in initial population **do**
        Calculate fitness function (makespan)
    end for
    Select S_i with minimum makespan
    Apply crossover on selected solutions (DMG)
    Send them to next generation.
    Exit when S_i with minimum makespan is found
end while

9.2.4 Illustrative Example

This section will describe the implementation of our work by using an illustrative example.

9.2.5 Creation of FDQs

Our example considers the FDQs which are time-sensitive in nature and have stringent timing deadlines. The designed FDQs are executed over the database created in an SQL server. This database contains 85,000 values from sensors and actuators. These FDQs are used to detect two types of faults (i) fault in heater damper and (ii) concentration level of CO_2 and temperature for fire detection. Therefore all of these FDQs have stringent timing deadlines provided by the system. The FDQs mentioned in Table 9.6 are created for finding the fire in the NR_1 considering that the HeaterWarningCounter has a value equal to zero and Heatvalue and $CO_2 concentration$ values are above threshold.

Table 9.6: FDQ_1 for Fire Detection CR_1 Floor 1 (F_1)

Names	FDQS
CQ_1	Select HeaterWarningCounter from Heateractuator where HeaterWarningCounter=0
CQ_2	Select TemperatureValue, HeaterWarningCounter from TemperatureSensor T, Heateractuator H where T.id=H.id and TemperatureValue>1100 and HeaterWarningCounter=0
CQ_3	Select $CO_2 concentration$, TemperatureValue from $CO_2 Sensor$ C, TemperatureSensor T where C.id=T.id and $CO_2 concentration > 12800$ and HeaterWarningCounter=10
CQ_4	Select Heatvalue $CO_2 Sensorvalue$, TemperatureValue from HeatSensor H, $CO_2 Sensor$ C, TemperatureSensor T where H.id=C.id and C.id=T.id and Heatvalue>1100 and $CO_2 concentration > 12800$ and HeaterWarningCounter=0

9.2.6 Creation of FDQs after applying Cross join and changing Join Order

In this step, the cross join method is applied to the FDQs generated in Table 9.6 as mentioned in Algorithm 5 (see Line 3-7). The new FDQs after applying a cross join method and different join orders are shown in Table 9.7.

Table 9.7: FDQ_2: Application of JM and JO to FDQ_1 mentioned in Table 9.6

Names	FDQs
CQ_5	Select HeaterWarningCounter from Heateractuator where HeaterWarningCounter=0
CQ_6	Select TemperatureValue, HeaterWarningCounter from TemperatureSensor T cross join Heateractuator H where T.id=H.id and TemperatureValue>1100 and HeaterWarningCounter=0
CQ_7	Select $CO_2Sensorvalue$, TemperatureValue from $CO_2concentration$ C cross join TemperatureSensor T on C.id=T.id where $CO_2concentration > 12800$ and HeaterWarningCounter=0
CQ_8	Select Heatvalue $CO_2Sensorvalue$, TemperatureValue from ((HeatSensor H cross join $CO_2Sensor$ C) cross join TemperatureSensor T) where H.id=C.id and C.id=T.id and Heatvalue>1100 and $CO_2concentration > 12800$ and HeaterWarningCounter=0

Creation of DMG for the Search Space

After the creation of FDQs shown in the previous step, our next step is the creation of the DMG. These DMGs are created based on FDQs created in Table 9.6 and 9.7. These DMGs are then added to the search space of our GA. Similarly, different DMGs are created by applying different join methods and join orders to FDQs (cf. Algorithm 5 Line 1). The DMG created from FDQ_1 (cf. Table 9.6) is named as DMG_1. The DMG created from FDQ_2 after applying the cross join method and selecting the best join order (cf. Table 9.7) is named as DMG_2. Both DMGs are represented in Fig. 9.4.

9.3 Genetic Algorithm Based Optimization of the DMG

This section will elaborate the implementation of our proposed GA using an illustrative example.

9.3.1 Initial Population

Each DMG that is created for the catastrophic room from the FDQ Table (cf. Table 9.6, Table 9.7) is considered to be the initial population for our search space. The chromosome representation of DMGs is shown in Table 9.8.

Table 9.8: Chromosome Representation for DMGs

Query Type	Representation
Select Queries	1
Equi-join (2 relations)	2
Equi-join (3 relations)	3
Cross join (2 relations)	4
Cross join (3 relations)	5
Loop join (2 relations)	6
Loop join (3 relations)	7
Hash join (2 relations)	8
Hash join (3 relations)	9

9.3.2 Fitness Function

As mentioned in Section 9.2 for each DMG we have to calculate the makespan. To calculate the makespan we require two parameters (i). WCET, (ii). W_{ij} (data transfer from parent to child node). These two parameters are given as an input to the scheduler for calculating the makespan. Each FDQ is run thousands of times to derive its minimum and maximum execution time. Then a safety margin is added to the WCET of FDQs to satisfy the timing constraints of the system [129]. WCET of the FDQ CQ_1 is shown in Fig. 9.5. The amount of data weight W_i transferred between parent and child node is calculated by Eq. 9.1. The input given to the scheduler for calculating the makespan is shown in Table 9.9.

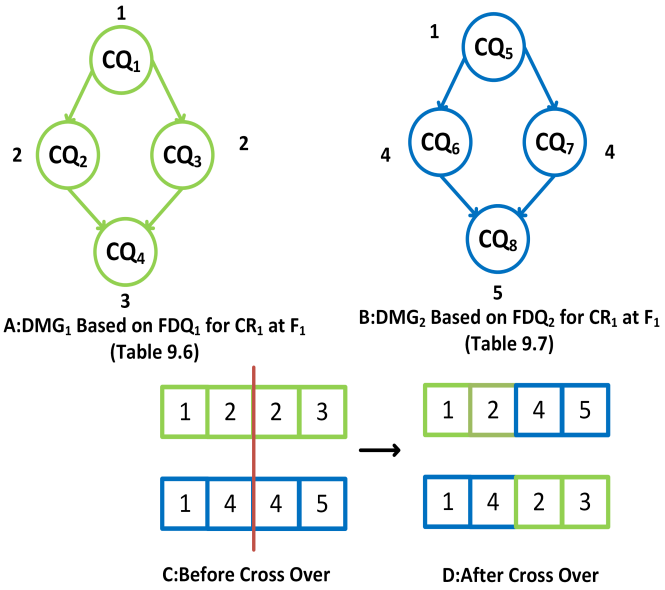


Figure 9.4: Chromosome and Crossover of DMG

$$W_{ij} = [rowsize] \times [noofrows] \tag{9.1}$$

Table 9.9: Input for Scheduler for calculation of makespan Fig. 9.4

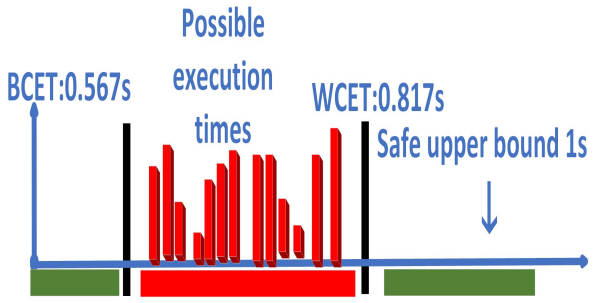
FDQ	WCET (Secs)	W (MB)
CQ_1	1.817	
CQ_2	2.541	W_1: 0.769
CQ_3	2.544	W_2: 0.769
CQ_4	3.922	W_3, W_4 : 2.769
Makespan (DMG_1)	7.189	

9.3.3 Selection

In this step, the fittest solution is selected and passed to the next generation. For each solution (DMG) the fitness score is measured by calculating its makespan. The solutions that have minimum makespan are considered to be the fittest and sent to the next generations.

9.3.4 Cross Over

We have applied one point cross over in our implementation. Therefore after the implementation of cross over if we get the similar DMG then these DMGs are not sent to the next generation. The cross over operation is shown in Fig. 9.4. We have not applied the mutation to our algorithm because we cannot change the order of the nodes in the DMG otherwise the parent-child relationship becomes irrelevant and wrong.



BCET:0.567 WCET with safe upper bound:1.817

Figure 9.5: WCET of CQ_1 from Table 9.6

9.4 Results

This section presents the results of our work. In our experimental results we have taken three types of network topologies including (i) star, (ii) ring and (iii) bus. We have created different DMGs with varying sizes. The size of our DMG ranges between 10 to 80 nodes (building comprised of 10 to 80 rooms). We also considered different numbers of processors while calculating the makespan of our DMG. We can get different results when we increase the number of processors in all topologies. All the results are generated by creating the DMGs based on the safety-critical queries (example mentioned in Table 9.6).

9.4.1 Result 1 and Result 2

In our first result we have considered the two parameters including (i) star topology and (ii) four end processors (cf. Fig. 9.6A). In our second result, we have considered a similar topology with six processors (cf. Fig. 9.6B).

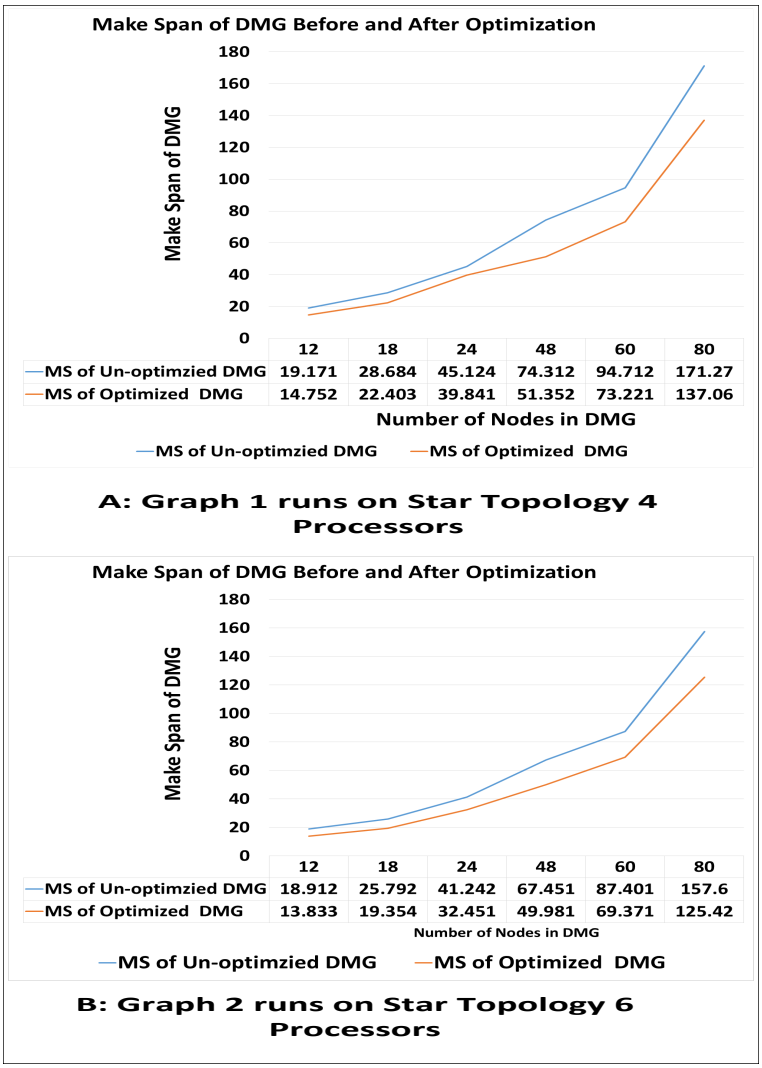


Figure 9.6: Makespan of DMG for Star Topology

9.4.2 Result 3 and Result 4

In our third result we have considered the same parameters including (i) bus topology and (ii) four processors (Fig. 9.7C). In our fourth result we have considered a similar topology

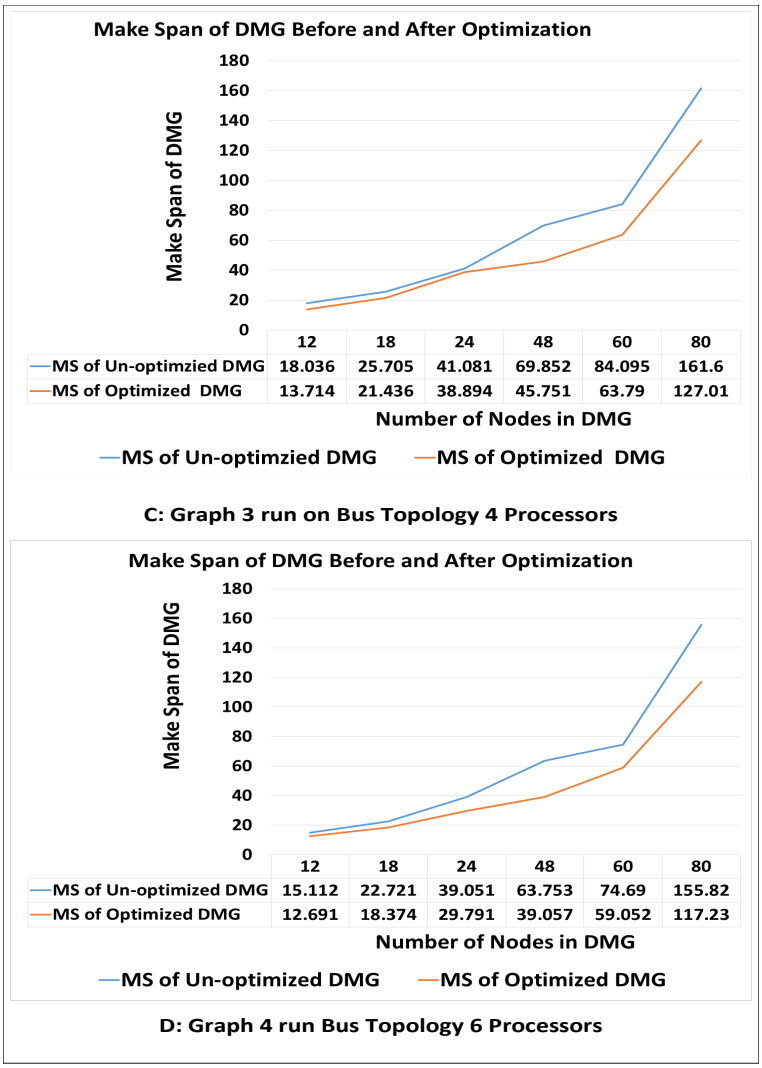


Figure 9.7: Makespan of DMG for Bus Topology

with six processors (Fig. 9.7D).

9.4.3 Result 5 and Result 6

In our third result we have considered the parameters including (i) ring topology and (ii) four end processors (Fig. 9.8E). In our fourth result we have considered a similar topology with six processors (Fig. 9.8F).

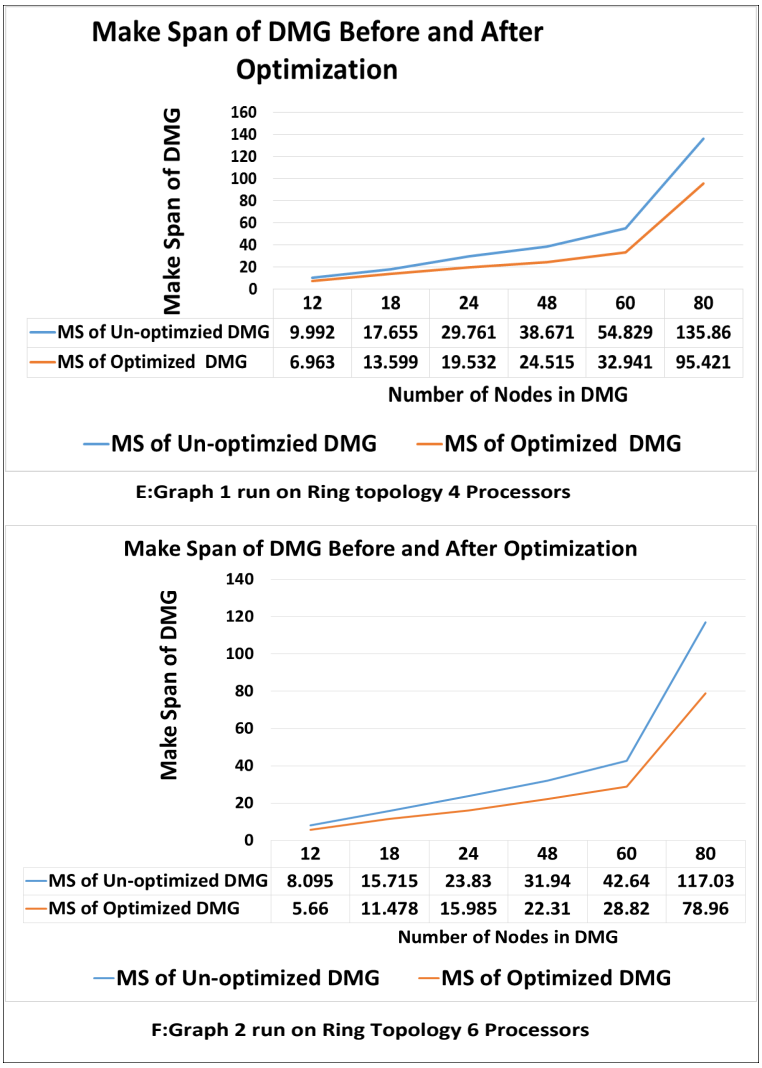


Figure 9.8: Makespan of DMG for Ring Topology

9.4.4 Conclusion About Results

If we compare the makespan calculation of all the DMGs with different topologies, it is evident that when the number of processors increases the overall makespan of the DMG decreases, especially in case of bigger DMGs. Therefore in our case, the ring topology gives the best result in terms of minimization of makespan of DMGs. Since there are more than one connection between the processors in the ring topology, therefore there is more slots for message transfere in comparison to star and bus topology. So the message does not have to wait for the link to be free which in turn reduces the makespan of the proposed

DMG.

9.5 Conclusion

Modern HVAC systems are used both for maintaining comfortable environment levels and monitoring the occurrence of catastrophic situations e.g. detection of a fire. By applying the query optimization techniques from the previous chapters we presented a query-based fault detection and diagnosis technique in HVAC systems to find faults in both safety-critical and non-safety critical scenarios. We considered the occurrence of a fire in the building as a safety-critical and time-sensitive situation, i.e. the faults in the sensors/actuators in the HVAC systems need to be identified and rectified before the system's defined time-bound. The normal operation of the HVAC system does not have such time-bounds and the faults in this case are diagnosed to reduce the energy consumption by the system. We have used diagnostic fault queries that are realized on the features and symptoms extracted from the sensors in the HVAC systems. Using these diagnostic queries, we formulated a DMG for the timely analysis of the faults. To ensure that we meet the stringent timing constraints of the system, we optimize the DMG using a GA and then give it to a scheduler. The scheduler generates an optimal execution plan for the DMG and gives the time at which the fault will be detected, i.e. execution time of the DMG. We ran our fault detection technique on buildings with 10, 20, 40, 60, and 80 rooms, respectively. Our results show that our technique minimize the makespan of the DMG which enables the process of fault detection and diagnosis to be completed within the stringent timing deadline of the system.

CHAPTER 10

CONCLUSION

This chapter gives a brief summary of the overall results presented in the context of our research. Different techniques are introduced for minimizing the makespan of a DMG so that the timing bound of an embedded system is fulfilled. These techniques are tested on two different domains including (i) vehicles and (ii) HVAC systems.

For the implementation of our research work, we have designed our own application and architecture models. Our application model known as Diagnostic Multi-Query Graphs (DMG) is a directed acyclic graph which contains the Fault Diagnostic Queries (FDQs). Each node of the DMG contains a FDQ which is executed over the database created in Pervasive SQL. Each FDQ has a worst-case execution time which is determined by using measurement-based method(i.e., measuring the maximum execution time of each FDQ). For measuring the makespan of the DMG, the WCET of the FDQ (within the node of the DMG) along with the Weight (W) of the edge (number of tuples that are transferred between parent and child node) is given to the scheduler. The scheduler calculates the overall time taken by the DMG for its execution.

The first method, which extracts fault diagnostic queries from a DMG and applies a class based categorization. The class based categorization technique selects the best access method for queries based on the query type. After the selection of the access method, the best join order is selected by calculating the selectivity factor. The search space for join orders has been minimized by implementing the left depth tree. The presented results show that the overall worst case execution time decreases up to 30% after the query optimization.

The second optimization technique is based on graph pruning and query optimization so that the overall makespan of a DMG can be minimized. The optimization is split into two steps. The first step comprises the pruning of the graph nodes without affecting the

semantics of diagnostic queries. Each graph node that satisfies a certain set of constraints is deleted and its query is merged with its neighborhood nodes. The constraints for pruning and merging are based on the matching of SQL operations (select or join) and the data tables between the queries. The new graph generated after pruning is a subset of the original graph based on the merged FDQs from the deleted nodes. The second step is based on the optimization of the fault diagnostic queries in each node of the DMG, by selecting the best query execution plan. After the DMG is pruned and queries are optimized the new DMG is given as an input to a scheduler to determine the ensuing makespan.

The third optimization technique meets two objectives. The first objective is the minimization of the makespan in order to fulfill the timing constraints of real time systems. The second objective is to decrease the utilization of CPU. The first goal is achieved by applying per table query aware partitioning and join aware partitioning on the FDQs. The best QEPs with minimum worst case execution time are selected for the purpose of optimization. The second objective of decreasing resource consumption is achieved by applying the concept of history intervals and skip values. The history interval specifies from which execution of a parent node, a child node requires data to complete its processing. In order to decrease the data communication overhead, a skip value S is introduced. Results show that due to the optimization techniques, the overall makespan of the DMG in the context of select and join based FDQs is decreased.

The forth presented technique for minimizing the make span of task graphs in real-time systems is based on a genetic algorithm (GA). Different task graphs on the basis of FDQ trees are generated. These task graphs are considered as a solution for our search space in the GA. The task graph with minimum make span is selected by our proposed GA, so that the deadline constraint of the system can be fulfilled. Different task graphs are tested with different topologies. The results in the context of bushy trees over a ring topology are better as compares to left deep trees. The overall makespan of the diagnostic query is reduced by almost 60% after the optimization is applied.

The last presented technique is tested on HVAC systems. For the detection of hazardous situations, e.g. a fire, defective components in the HVAC systems may hide the occurrence of a catastrophe or cause stressful situations with false alarms. In both cases, there is a possibility of damage to human life and property that can be prevented with working detectors. In this chapter, we propose a diagnostic query-based fault detection and diagnosis technique to monitor faults in the HVAC systems in safety-critical and non safety-critical situations. We consider the critical situations (detection of fire) time-sensitive and our technique determines whether an actual disaster has occurred or the system itself has malfunctioned within the system's defined time-bound. Our technique ensures that the HVAC system gives a time-critical and reliable response for detecting a disaster in the building. Fault detection in non-critical conditions has no time restriction and is performed to minimize the energy consumption of the HVAC systems. We have used fault diagnostic queries that are realized on the features and symptoms extracted from the sensors in the HVAC system. Using these diagnostic queries, we formulate a diagnostic multi-query graph (DMG) for the structured and timely analysis of faults. To ensure that we meet the stringent timing constraints of the system, we optimize the DMG using a genetic algorithm. The optimized DMG serves as the input for the scheduler that gives the execution time for fault detection and generates an optimal plan for the execution of the DMG.

Minimization of the makespans enables the completion of the process of fault detection and diagnosis within the time bound provided by the system. Therefore, introducing the process of optimization in active diagnosis increases the overall reliability of the system, and it is ensured that the system is capable of detecting a fault before any catastrophic situation may occur.

REFERENCES

[1] H. Kopetz, "Internet of things," in *Real-time systems*, Springer, 2011, pp. 307–323.

[2] M. Nahas, "Bridging the gap between scheduling algorithms and scheduler implementations in time-triggered embedded systems.," PhD thesis, University of Leicester, 2009.

[3] B. Sun, X. Li, B. Wan, C. Wang, X. Zhou, and X. Chen, "Definitions of predictability for cyber physical systems," *Journal of Systems Architecture*, vol. 63, pp. 48–60, 2016.

[4] M. Nahas and A. M. Nahhas, "Ways for implementing highly-predictable embedded systems using time-triggered co-operative (ttc) architectures," *Embedded Systems-Theory and Design Methodology*, 2012.

[5] T. Abdelzaher, C. Gill, R Rajkumar, and J. Stankovic, "Distributed real-time and embedded systems research in the context of geni," Technical report, National Science Foundation Workshop, 2006.

[6] A.-M. Grisogono, "The implications of complex adaptive systems theory for c2," DEFENCE SCIENCE and TECHNOLOGY ORGANISATION EDINBURGH (AUSTRALIA) LAND . . ., Tech. Rep., 2006.

[7] G. J. Ducard, *Fault-tolerant flight control and guidance systems: Practical methods for small unmanned aerial vehicles*. Springer Science & Business Media, 2009.

[8] X. Mingyao and L. Xiongfei, "Embedded database query optimization algorithm based on particle swarm optimization," in *Measuring Technology and Mechatronics Automation (ICMTMA), 2015 Seventh International Conference on*, IEEE, 2015, pp. 429–432.

[9] N. Kandasamy, J. P. Hayes, and B. T. Murray, "Time-constrained failure diagnosis in distributed embedded systems: Application to actuator diagnosis," *IEEE Transactions on parallel and distributed systems*, vol. 16, no. 3, pp. 258–270, 2005.

[10] Y. Zhang and J. Jiang, "Bibliographical review on reconfigurable fault-tolerant control systems," *IFAC Proceedings Volumes*, vol. 36, no. 5, pp. 257–268, 2003.

[11] L.-L. Huang and A. Shimizu, "A multi-expert approach for robust face detection," *Pattern Recognition*, vol. 39, no. 9, pp. 1695–1703, 2006.

[12] S. Skogestad, "Self-optimizing control: The missing link between steady-state optimization and control," *Computers & Chemical Engineering*, vol. 24, no. 2-7, pp. 569–575, 2000.

[13] Y Chetouani, "Fault detection in a chemical reactor by using the standardized innovation," *Process Safety and Environmental Protection*, vol. 84, no. 1, pp. 27–32, 2006.

[14] G.-C. Luh and W.-C. Cheng, "Immune model-based fault diagnosis," *Mathematics and Computers in Simulation*, vol. 67, no. 6, pp. 515–539, 2005.

[15] S. Dash and V. Venkatasubramanian, "Challenges in the industrial applications of fault diagnostic systems," *Computers & chemical engineering*, vol. 24, no. 2-7, pp. 785–791, 2000.

[16] V. Venkatasubramanian, R. Rengaswamy, and S. N. Kavuri, "A review of process fault detection and diagnosis: Part ii: Qualitative models and search strategies," *Computers & chemical engineering*, vol. 27, no. 3, pp. 313–326, 2003.

[17] Actian, *Database management, integration and analytics*, 2001 (accessed February 3, 2019).

[18] D. Nyström, A. Tešanovic, M. Nolin, C. Norström, and J. Hansson, "Comet: A component-based real-time database for automotive systems," in *Proceedings of the workshop on software engineering for automotive systems*, IET, 2004, pp. 1–8.

[19] Oracle, *Timesten: Fastest oltp database, ultra high availability, elastic scalability*, 2009 (accessed February 3, 2019).

[20] W. Kang, S. H. Son, and J. A. Stankovic, "Design, implementation, and evaluation of a qos-aware real-time embedded database," *IEEE Transactions on Computers*, vol. 61, no. 1, pp. 45–59, 2010.

[21] S. Andler, J Hansson, J Mellin, J Eriksson, and B Eftring, "An overview of the deeds real-time database architecture," in *Proceedings of the Sixth International Workshop on Parallel and Distributed Real-Time Systems*, Citeseer, 1998.

[22] S. G. Perlman and R. van der Laan, *System for streaming databases serving real-time applications used through streaming interactive*, US Patent 8,834,274, 2014.

[23] F. P. Preparata, G. Metze, and R. T. Chien, "On the connection assignment problem of diagnosable systems," *IEEE Transactions on Electronic Computers*, no. 6, pp. 848–854, 1967.

[24] E. A. Alchieri, A. N. Bessani, J. da Silva Fraga, and F. Greve, "Byzantine consensus with unknown participants," in *International Conference On Principles Of Distributed Systems*, Springer, 2008, pp. 22–40.

[25] J. Chen and R. J. Patton, *Robust model-based fault diagnosis for dynamic systems*. Springer Science & Business Media, 2012, vol. 3.

[26] L. Portinale and D. Codetta-Raiteri, "Using dynamic decision networks and extended fault trees for autonomous fdir," in *2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, IEEE, 2011, pp. 480–484.

[27] T. Jiang, K. Khorasani, and S. Tafazoli, "Parameter estimation-based fault detection, isolation and recovery for nonlinear satellite models," *IEEE Transactions on control systems technology*, vol. 16, no. 4, pp. 799–808, 2008.

[28] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*. Springer Science & Business Media, 2011.

[29] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*, IEEE, 1998, pp. 4–13.

[30] M. Wei, J. Liu, T. Li, X. Xu, W. Hu, and D. Zhao, "Fault-tolerant scheduling of real-time tasks on heterogeneous systems," in *2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, IEEE, 2017, pp. 1006–1011.

[31] W. M. Elseaidy, "Safety and reliability of real-time engineering systems using formal methods," 1995.

[32] J. A. Stankovic and K. Ramamritham, *What is predictability for real-time systems?* 1990.

[33] D. Fedasyuk, R. Chopey, and B. Knysh, "Architecture of a tool for automated testing the worst-case execution time of real-time embedded systems' firmware," in *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, IEEE, 2017, pp. 278–281.

[34] A. R. Conway, M. J. Kane, M. F. Bunting, D. Z. Hambrick, O. Wilhelm, and R. W. Engle, "Working memory span tasks: A methodological review and user's guide," *Psychonomic bulletin & review*, vol. 12, no. 5, pp. 769–786, 2005.

[35] S. Einspieler, B. Steinwender, and W. Elmenreich, "Integrating time-triggered and event-triggered traffic in a hard real-time system," in *2018 IEEE Industrial Cyber-Physical Systems (ICPS)*, IEEE, 2018, pp. 122–128.

[36] J. Huang, J. Voeten, O. Florescu, P. Van Der Putten, and H. Corporaal, "Predictability in real-time system development," in *Advances in Design and Specification Languages for SoCs*, Springer, 2005, pp. 123–139.

[37] A. Nogueira and M. Calha, "Predictability and efficiency in contemporary hard rtos for multiprocessor systems," in *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, vol. 2, 2011, pp. 3–8.

[38] W. Ko, J. Yoo, I. Kang, J. Jun, and S.-S. Lim, "Lightweight, predictable hypervisor for arm-based embedded systems," in *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, IEEE, 2016, pp. 109–109.

[39] J Krumm, *Ubiquitous computing fundamentals, microsoft corporation*, 2010.

[40] W. Khan, W. Ahmad, B. Luo, and E. Ahmed, "Sql database with physical database tuning technique and nosql graph database comparisons," in *2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, IEEE, 2019, pp. 110–116.

[41] T. M. Connolly and C. E. Begg, *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.

[42] A. Dennis, B. H. Wixom, and D. Tegarden, *Systems Analysis and Design UML Version 2.0*. Wiley, 2009.

[43] B. J. Bulkowski and V. Srinivasan, *Real-time transaction scheduling in a distributed database*, US Patent 8,799,248, 2014.

[44] A. Bestavros and V. Fay-Wolfe, *Real-time database and information systems: research advances: research advances*. Springer Science & Business Media, 2012, vol. 420.

[45] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees, "The sap hana database–an architecture overview.," *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 28–33, 2012.

[46] Mcobject, *Professionals development database*, 2002 (accessed August 3, 2019).

[47] L. Corporation, *Real time databases*, 2009 (accessed August 3, 2019).

[48] L. M. Corporation, *Real time databases*, 2009.

[49] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann, *et al.*, "T-crest: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.

[50] J. Yan and W. Zhang, "Wcet analysis for multi-core processors with shared l2 instruction caches," in *2008 IEEE Real-Time and Embedded Technology and Applications Symposium*, IEEE, 2008, pp. 80–89.

[51] D. Sangorrín, M. G. Harbour, H. Pérez, and J. J. Gutiérrez, "Managing transactions in flexible distributed real-time systems," in *International Conference on Reliable Software Technologies*, Springer, 2010, pp. 251–264.

[52] L. X. Yan, Z. Liang, Z. Jing, and F. X. Dong, "Resource isolation policy for task scheduling strategy in open real-time systems," in *Proceedings of the 31st Chinese Control Conference*, IEEE, 2012, pp. 2374–2379.

[53] A. Hangan and G. Sebestyen, "Cyclic executive-based method for scheduling hard real-time transactions on distributed systems," in *2011 IEEE 7th International Conference on Intelligent Computer Communication and Processing*, IEEE, 2011, pp. 441–444.

[54] G. Lohman, "Is query optimization a "solved" problem," in *Proc. Workshop on Database Query Optimization*, Oregon Graduate Center Comp. Sci. Tech. Rep, vol. 13, 2014.

[55] M. Khan and M. Khan, "Exploring query optimization techniques in relational databases," *International Journal of Database Theory & Application*, vol. 6, no. 3, pp. 11–20, 2013.

[56] N. Kumari, "Sql server query optimization techniques—tips for writing efficient and faster queries," *International Journal of Scientific and Research Publications*, vol. 2, no. 6, pp. 1–4, 2012.

[57] P. M. Pardalos, D.-Z. Du, and R. L. Graham, *Handbook of combinatorial optimization*. Springer, 2013.

[58] V. K. Myalapalli, T. P. Totakura, and S. Geloth, "Augmenting database performance via sql tuning," in *Energy Systems and Applications, 2015 International Conference on*, IEEE, 2015, pp. 13–18.

[59] B. Plale and K. Schwan, "Optimizations enabled by a relational data model view to querying data stream," in *null*, IEEE, 2001, 10020a.

[60] ——, "Dynamic querying of streaming data with the dquob system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, no. 4, pp. 422–432, 2003.

[61] A. de Castro Alves, A. Srinivasan, and M. James, *Handling faults in a continuous event processing (cep) system*, US Patent 9,262,258, 2016.

[62] H. Yinghua, M. Yanchun, and Z. Dongfang, "The multi-join query optimization for smart grid data," in *Intelligent Computation Technology and Automation (ICICTA), 2015 8th International Conference on*, IEEE, 2015, pp. 1004–1007.

[63] W. C. Eidson and J. Collins, *Methods and systems for joining indexes for query optimization in a multi-tenant database*, US Patent 8,706,715, 2014.

[64] V. K. Myalapalli and A. Chakravarthy, "Revamping sql queries for cost based optimization," *Unpublished*,

[65] A. K. Z. Al Saedi, M. B. M. Deris, *et al.*, "An efficient multi join query optimization for dbms using swarm intelligent approach," in *Information and Communication Technologies (WICT), 2014 Fourth World Congress on*, IEEE, 2014, pp. 113–117.

[66] D. Petkovic, "Comparison of different solutions for solving the optimization problem of large join queries," in *Advances in Databases Knowledge and Data Applications (DBKDA), 2010 Second International Conference on*, IEEE, 2010, pp. 51–55.

[67] Q. Wen, S. Yi, J. Tian, and Q. Li, "A muti-join query optimization for the aerospace data," in *Engineering and Industries (ICEI), 2011 International Conference on*, IEEE, 2011, pp. 1–7.

[68] A. Aljanaby, E. Abuelrub, and M. Odeh, "A survey of distributed query optimization.," *Int. Arab J. Inf. Technol.*, vol. 2, no. 1, pp. 48–57, 2005.

[69] R. Ghaemi, A. M. Fard, H. Tabatabaee, and M. Sadeghizadeh, "Evolutionary query optimization for heterogeneous distributed database systems," *World Academy of science*, vol. 43, pp. 43–49, 2008.

[70] S. M. Mahajan and V. P. Jadhav, "An analysis of execution plans in query optimization," in *Communication, Information & Computing Technology (ICCICT), 2012 International Conference on*, IEEE, 2012, pp. 1–5.

[71] Z. Zhenyou, J. Zhang, L. Shu, and C. Zhi, "Research on the integration and query optimization for the distributed heterogeneous database," in *Computer Science and Network Technology (ICCSNT), 2011 International Conference on*, IEEE, vol. 3, 2011, pp. 1533–1536.

[72] V Vaidehi and D. S. Devi, "Distributed database management and join of multiple data streams in wireless sensor network using querying techniques," in *Recent Trends in Information Technology (ICRTIT), 2011 International Conference on*, IEEE, 2011, pp. 594–599.

[73] I. Azari, "Efficient execution of query in distributed database systems," in *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, IEEE, vol. 4, 2010, pp. V4–428.

[74] W. Zheng, X. Jin, F. Deng, S. Mo, Y. Qu, Y. Yang, X. Li, S. Long, C. Zheng, J. Liu, *et al.*, "Database query optimization based on parallel ant colony algorithm," in *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)*, IEEE, 2018, pp. 653–656.

[75] F. Xiao and M. Aritsugi, "Nested pattern queries processing optimization over multi-dimensional event streams," in *Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual*, IEEE, 2013, pp. 74–83.

[76] J. B. Dugan, S. J. Bavuso, and M. A. Boyd, "Dynamic fault-tree models for fault-tolerant computer systems," *IEEE Transactions on reliability*, vol. 41, no. 3, pp. 363–377, 1992.

[77] M. R. Maurya, R Rengaswamy, and V Venkatasubramanian, "A signed directed graph and qualitative trend analysis-based framework for incipient fault diagnosis," *Chemical Engineering Research and Design*, vol. 85, no. 10, pp. 1407–1422, 2007.

[78] M. Van and H.-J. Kang, "Two-stage feature selection for bearing fault diagnosis based on dual-tree complex wavelet transform and empirical mode decomposition," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 230, no. 2, pp. 291–302, 2016.

[79] L. Duan, F. Wang, R. Guo, and R. Gai, "A fault diagnosis method for information systems based on weighted fault diagnosis tree," in *2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*, IEEE, 2017, pp. 1–6.

[80] H. Meyerhenke, B. Monien, and T. Sauerwald, "A new diffusion-based multilevel algorithm for computing graph partitions," *Journal of Parallel and Distributed Computing*, vol. 69, no. 9, pp. 750–761, 2009.

[81] R. Andersen, F. Chung, and K. Lang, "Local graph partitioning using pagerank vectors," in *null*, IEEE, 2006, pp. 475–486.

[82] K. Schloegel, G. Karypis, and V. Kumar, *Graph partitioning for high performance scientific simulations*. Army High Performance Computing Research Center, 2000.

[83] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proceedings of the 19th design automation conference*, IEEE Press, 1982, pp. 175–181.

[84] A. Pothen, H. D. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM journal on matrix analysis and applications*, vol. 11, no. 3, pp. 430–452, 1990.

[85] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[86] B. Suger, G. D. Tipaldi, L. Spinello, and W. Burgard, "An approach to solving large-scale slam problems with a small memory footprint," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, 2014, pp. 3632–3637.

[87] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao, "Efficient graph similarity search over large graph databases," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 4, pp. 964–978, 2015.

[88] F. Yang and D. Xiao, "Progress in root cause and fault propagation analysis of large-scale industrial processes," *Journal of Control Science and Engineering*, vol. 2012, 2012.

[89] N. Mehranbod, M. Soroush, M. Piovoso, and B. A. Ogunnaike, "Probabilistic model for sensor fault detection and identification," *AIChE Journal*, vol. 49, no. 7, pp. 1787–1802, 2003.

[90] S. Verron, J. Li, and T. Tiplica, "Fault detection and isolation of faults in a multivariate process with bayesian network," *Journal of Process Control*, vol. 20, no. 8, pp. 902–911, 2010.

[91] S. A. Arogeti, D. Wang, C. B. Low, and M. Yu, "Fault detection isolation and estimation in a vehicle steering system," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 12, pp. 4810–4820, 2012.

[92] F. Zhao, Q. Sun, J. Zhan, L. Nie, and Z. Xu, "The real-time database application in transformer substation hotspot monitoring system," in *16th International Conference on Advanced Communication Technology*, IEEE, 2014, pp. 941–944.

[93] G. J. Hahn and J. Packowski, "A perspective on applications of in-memory analytics in supply chain management," *Decision Support Systems*, vol. 76, pp. 45–52, 2015.

[94] O. Diallo, J. J. Rodrigues, and M. Sene, "Real-time data management on wireless sensor networks: A survey," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 1013–1021, 2012.

[95] L. Qiaoyu, L. Jianwei, and X. Yubin, "Performance analysis of data organization of the real-time memory database based on red-black tree," in *2010 International Conference on Computing, Control and Industrial Engineering*, IEEE, vol. 1, 2010, pp. 428–430.

[96] S. Mittal, "Power management techniques for data centers: A survey," *arXiv preprint arXiv:1404.6681*, 2014.

[97] Y. Zhou and K.-D. Kang, "A federated approach for increasing the timely throughput of real-time data services," in *2012 IEEE 18th Real Time and Embedded Technology and Applications Symposium*, IEEE, 2012, pp. 163–172.

[98] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "Deucon: Decentralized end-to-end utilization control for distributed real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 7, pp. 996–1009, 2007.

[99] N. Marz and J. Warren, *Big Data: Principles and best practices of scalable real-time data systems*. New York; Manning Publications Co., 2015.

[100] L. Kong and J. Jiang, "A safe measurement-based worst-case execution time estimation using automatic test-data generation," in *Dependable Computing (PRDC), 2010 IEEE 16th Pacific Rim International Symposium on*, IEEE, 2010, pp. 245–246.

[101] Y. Wang, "Research on the execution time analysis technology of the worst case system in real time system," in *Measuring Technology and Mechatronics Automation (ICMTMA), 2017 9th International Conference on*, IEEE, 2017, pp. 397–402.

[102] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner, "Measurement-based worst-case execution time analysis," in *Software Technologies for Future Embedded and Ubiquitous Systems, 2005. SEUS 2005. Third IEEE Workshop on*, IEEE, 2005, pp. 7–10.

[103] M. Mucha, J. Mottok, and M. Deubzer, "Probabilistic worst case response time estimation for multi-core real-time systems," in *Embedded Computing (MECO), 2015 4th Mediterranean Conference on*, IEEE, 2015, pp. 31–36.

[104] J. Rosén, P. Eles, Z. Peng, and A. Andrei, "Predictable worst-case execution time analysis for multiprocessor systems-on-chip," in *Electronic Design, Test and Application (DELTA), 2011 Sixth IEEE International Symposium on*, IEEE, 2011, pp. 99–104.

[105] J. Rosén, C.-F. Neikter, P. Eles, Z. Peng, P. Burgio, and L. Benini, "Bus access design for combined worst and average case execution time optimization of predictable real-time applications on multiprocessor systems-on-chip," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*, IEEE, 2011, pp. 291–301.

[106] M. Lindgren, H. Hansson, and H. Thane, "Using measurements to derive the worst-case execution time," in *rtcsa*, IEEE, 2000, p. 15.

[107] J. Kim, H. Oh, H. Ha, S.-H. Kang, J. Choi, and S. Ha, "An ilp-based worst-case performance analysis technique for distributed real-time embedded systems," in *2012 IEEE 33rd Real-Time Systems Symposium*, IEEE, 2012, pp. 363–372.

[108] G. Nelissen, H. Carvalho, D. Pereira, and E. Tovar, "Demo abstract: Run-time monitoring environments for real-time and safety critical systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*, IEEE, 2016, pp. 1–1.

[109] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis–the symta/s approach," *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[110] S. Quinton, M. Hanke, and R. Ernst, "Formal analysis of sporadic overload in real-time systems," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2012, pp. 515–520.

[111] Z. A. Hammadeh, S. Quinton, and R. Ernst, "Extending typical worst-case analysis using response-time dependencies to bound deadline misses," in *Embedded Software (EMSOFT), 2014 International Conference on*, IEEE, 2014, pp. 1–10.

[112] J. Mäki-Turja and M. Sjödin, "Response-time analysis for transactions with execution-time dependencies.," in *RTNS*, 2011, pp. 139–146.

[113] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli, "Using statistical methods to compute the probability distribution of message response time in controller area network," *IEEE Transactions on Industrial Informatics*, vol. 6, no. 4, pp. 678–691, 2010.

[114] G. Bernat, A. Colin, and S. M. Petters, "Wcet analysis of probabilistic hard real-time systems," in *null*, IEEE, 2002, p. 279.

[115] V. C. Lee, K.-Y. Lam, and B. Kao, "Priority scheduling of transactions in distributed real-time databases," *Real-Time Systems*, vol. 16, no. 1, pp. 31–62, 1999.

[116] A Munnich and G Farber, "Calculating worst-case execution times of transactions in databases for event-driven, hard real-time embedded systems," in *Database Engineering and Applications Symposium, 2000 International*, IEEE, 2000, pp. 149–157.

[117] P. Azhen, G. Ruifeng, W. Haotian, D. Changyi, and Z. Liaomo, "Adaptive real-time scheduling for mixed task sets based on total bandwidth server," in *Intelligent Computation Technology and Automation (ICICTA), 2017 10th International Conference on*, IEEE, 2017, pp. 11–15.

[118] J. Kim, H. Oh, J. Choi, H. Ha, and S. Ha, "A novel analytical method for worst case response time estimation of distributed embedded systems," in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, IEEE, 2013, pp. 1–10.

[119] R. Obermaisser, R. I. Sadat, and F. Weber, "Active diagnosis in distributed embedded systems based on the time-triggered execution of semantic web queries," in *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on*, IEEE, 2014, pp. 222–229.

[120] L. Cucu and Y. Sorel, "Non-preemptive multiprocessor scheduling for strict periodic systems with precedence constraints," in *Proceedings of the 23rd Annual Workshop of the UK Planning and Scheduling Special Interest Group, PLANSIG*, vol. 4, 2004.

[121] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[122] S. Amin and R. Obermaisser, "Time-triggered scheduling of query executions for active diagnosis in distributed real-time systems," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2017, pp. 1–9.

[123] J. Fan, M. Zhang, S. Kok, M. Lu, and B. C. Ooi, "Crowdop: Query optimization for declarative crowdsourcing systems," *IEEE transactions on Knowledge and Data Engineering*, vol. 27, no. 8, pp. 2078–2092, 2015.

[124] I Wenzel, R Kirner, B Rieder, and P Puschner, "" measurement-based worst-case execution time analysis"; vortrag: Ieee workshop on software technologies for future embedded systems, seattle, washington; 16.05. 2005-17.05. 2005; in:" proceedings of the third workshop on software technologies for future embedded and ubiquitous systems (seus)", ieee,(2005), isbn: 0-7695-2357-9; s. 7-10.,"

[125] L. Carley, *Engine air temperature sensor*, 2019.

[126] N. Polyzotis, "Selectivity-based partitioning: A divide-and-union paradigm for effective query optimization," in *Proceedings of the 14th ACM international conference on Information and knowledge management*, ACM, 2005, pp. 720–727.

[127] H. Herodotou, N. Borisov, and S. Babu, "Query optimization techniques for partitioned tables," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, ACM, 2011, pp. 49–60.

[128] E. Wu *et al.*, "Shinobi: Insert-aware partitioning and indexing techniques for skewed database workloads," PhD thesis, Massachusetts Institute of Technology, 2010.

[129] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, *et al.*, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, p. 36, 2008.

[130] U. D. of Energy. (2008). Energy efficiency trends in residential and commercial buildings, (visited on 09/19/2019).

[131] Y. Kim, T. Schmid, M. B. Srivastava, and Y. Wang, "Challenges in resource monitoring for residential spaces," in *Proceedings of the First ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings*, ACM, 2009, pp. 1–6.

[132] M. Dey, S. P. Rana, and S. Dudley, "Smart building creation in large scale hvac environments through automated fault detection and diagnosis," *Future Generation Computer Systems*, 2018.

[133] J. Weimer, S. A. Ahmadi, J. Araujo, F. M. Mele, D. Papale, I. Shames, H. Sandberg, and K. H. Johansson, "Active actuator fault detection and diagnostics in hvac systems," in *Proceedings of the fourth ACM workshop on embedded sensing systems for energy-efficiency in buildings*, ACM, 2012, pp. 107–114.

[134] S. Katipamula and M. R. Brambley, "Methods for fault detection, diagnostics, and prognostics for building systems—a review, part i," *Hvac&R Research*, vol. 11, no. 1, pp. 3–25, 2005.

[135] N. Djuric and V. Novakovic, "Review of possibilities and necessities for building lifetime commissioning," *Renewable and Sustainable Energy Reviews*, vol. 13, no. 2, pp. 486–492, 2009.

[136] N. Fernandez, M. R. Brambley, S. Katipamula, H. Cho, J. K. Goddard, and L. H. Dinh, "Self-correcting hvac controls project final report," Pacific Northwest National Lab.(PNNL), Richland, WA (United States), Tech. Rep., 2010.

[137] L. Jagemar, D. Olsson, and F Schmidt, "The epbd and continuous commissioning," *Project Report, Building EQ, EIE/06/038/SI2*, vol. 448300, 2007.

[138] S. W. Kramer and P. Fleck, "Maintaining building function during a fire event: Analysis of hospital fire and smoke control systems," 2018.

[139] D. Sloo, N. U. Webb, E. J. Fisher, Y. Matsuoka, A. Fadell, and M. Rogers, *Smart-home control system providing hvac system dependent responses to hazard detection events*, US Patent 9,905,122, 2018.

[140] A. Behravan, N. Tabassam, O. Al-Najjar, and R. Obermaisser, "Composability modeling for the use case of demand-controlled ventilation and heating system," in *2019 6th International Conference on Control, Decision and Information Technologies (CoDIT)*, IEEE, 2019, pp. 1998–2003.