Minimizing the Make Span of Diagnostic Multi-Query Graphs Using Graph Pruning and Query Merging

Nadra Tabassam and Roman Obermaisser
University of Siegen, 57068 Siegen, Germany
nadra.tabassam@uni-siegen.de

Abstract— **Active diagnosis can significantly increase the reliability of a real-time system in case of fault occurrences. Root causes are identified for observed failures and the root causes are associated with suitable recovery actions such as the migration of services to spare resources or application-specific reconfiguration. Real-time databases and diagnostic multi-query graphs (DMG) are a promising technique for root-cause analysis. However, in order to ensure safety the completion of the diagnostic queries must be performed within strict timing bounds dictated by the environment. This paper presents optimization techniques for diagnostic multi-query graphs in order to minimize the make span of a root cause analysis. 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 queries from the deleted nodes. The second step is based on the optimization of the 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 make span.**

I. INTRODUCTION

Real time systems are different from regular systems in the sense that they are sensitive about the deadline of each task during execution. For example, the radar surveillance system is an example application which handles large amount of data with strict timing requirements. These systems detects the radar signatures and aircraft images and matches this data with the database of known images. The results based on these matchings are then used to make further decisions like selection of combat strategies [1]. These types of applications demand a high level reliability so that all the tasks in the system may complete within the predefine deadline of the system [2].

Therefore it can be notice that real time systems span a broad spectrum of complexity from very simple micro-controllers to highly sophisticated, complex and distributed systems. Due to this dimension of real time systems it is clear that these systems are managing the huge amounts of data coming from the different components. This data needs to be stored, analyzed and processed effectively during the active diagnosis in real time systems. In order to fulfill these requirements multiple real time database management systems have been designed which have properties like concurrency control, temporal consistency and timeliness [3].

Considering the above two aspects it is noticeable that these system have different types of diagnostic information to be processed. Due to this reason the overall size of task graph and database storing all the diagnostic information is immensely increased. Bigger task graphs 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 task graph in a manner that it may determine the fault in the system within the timing constraints defined by the system.

The major objective of our work is to minimize the make span of our (DMG) so that the overall timing deadlines of the system can fulfilled. An optimized task graph which is Diagnostic Multi-Query Graph (DMG) in our context, with a lower number of processing nodes and an optimized Query Execution Plan (QEP) is more likely to be schedule able and detect the faults within the given deadline.

Contributions: The major implementations we perform for achieving our objective are:

- Nodes of the DMG are pruned based on certain sets of constraints. Fault queries within the pruned nodes are not deleted but they are merged with their neighborhood nodes.
- The data that is being transfered (from parent node) to multiple deleted nodes is also transfered to the node which has the final merged query. In this way the data transference cost is also minimized as now the parent is transferring data to only one child instead of multiple children.
- After the graph nodes of the DMG are pruned each fault query is optimized by selecting the best access method and the best join order depending on the type of query (select or join). On the basis of these optimizations the best QEP with minimum cost in terms of time is selected.
- In order to minimize the data processing for child nodes, after each query execution a new data table is generated based on the new resultant tuples. In this way the data tuples which are not required by the child nodes are not transfered from their parent nodes.
- For running the diagnostic queries, an example database comprised of datasets from car sensors is designed in

Pervasive SQL (PSQL). PSQL is selected as a database in order to get benefits of a real time data management system. Database tables are populated when the insert queries are executed on the data that is generated from car sensors.

- The optimized DMG along with the Worst Case Execution Time (WCET) of each diagnostic query is provided as an input to a scheduler in order to calculate the make span of the DMG. For each query, the QEP with the maximum cost is also generated. The time taken by the worst QEP is considered WCET of each query [4].

II. Related Work

Graph optimization using partition has a broad spectrum. In order to get the broader overview of this research problem reader can refer to [5][6] The basic step for effectively processing the graph is to partition the graph and distribute the subgraphs on multiple nodes for parallel computing. Already existing partition strategies ignore the application features due to which the partition do not satisfy the needs of specific application. This problem is solved by considering Super block partition strategy in [7].

Google proposed Pregel, a system that adopts Bulk Synchronous Parallel model (BSP model) and processes the data of large scale graph iteratively. The Pregel system proposed new idea on how to process the large scale graphs effectively by breaking a large scale graph into small partitions[8].

In order to improve the quality of partitions in the graph a new technique based on the Cat Swarm Optimization (CSO) is introduced. This technique is based on the optimization with the Cauchy mutation and the Inertia weight [9]. A powerful perturbation-based tabu search algorithm is introduced which integrates a new multi-parent crossover operator. Experimental results shows that proposed approach performs much better than the already existing graph partitioning algorithm in terms of solution quality, specially in case where time limit ranges from several minutes to several hours [10].

Previously define graph partition technique do not work well for complex networks. This problem can be solved by parallelizing and adapting the label propagation technique which is originally developed for the purpose of graph clustering [11]. Large graphs are partitioned randomly by roughing and then the division is reverted by applying anti-roughening and optimization.

In [12] a graph optimization algorithm is proposed which enables the optimization of large graphs and reduces the 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, original graphs are optimized with small consumption of memory. According to our knowledge graph optimization based on the node pruning and query merging in real time systems was not done before.

III. Description of System Model

This section describes the overall components of our real time system. Fig.1 presents the work flow and basic building blocks of system.

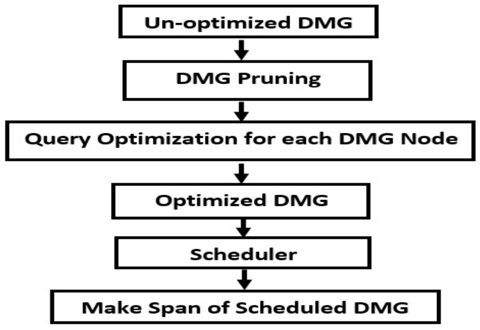
Un-optimized DMG
DMG Pruning
Query Optimization for each DMG Node
Optimized DMG
Scheduler
Make Span of Scheduled DMG

Fig. 1: Steps in Proposed System

A. System Model

Our proposed real time system is distributed and homogeneous in nature with strict timing constraints. This system is comprised of identical cores, communication links and routers. The cores are designated for the processing of query nodes of the DMG and they are not involve in any communication.

B. Directed Multi-Query Graph

- The basic building block of the system is Directed multi-query graph (DMG). Our proposed DMG stores the fault queries represented in the SQL format comprising of SQL operations select, insert and join. Each DMG is comprised of nodes and edges. The overall DMG is defined as

$$DMG = (N, E) \tag{1}$$

where

$$DMG(N) = (n_1, n_2, ..., n_i) \tag{2}$$

$$DMG(E) = (e_1, e_2, ..., e_i) \tag{3}$$

- Each graph node in the DMG represents the diagnostic query denoted by $Q(N)$. This query is required for the identification of faults in the system. Each edge E in the DMG represents a relationship between these queries.

$$DMG(e_i) = (n_p \to n_c) \tag{4}$$

Where a parent node n_p has a precedence over a child node n_c.

- Diagnostic fault queries are based on the features and symptoms extracted directly from the sensors data.
- Time Period: As we are dealing with periodic queries so each graph node $n \in N$ will repeat its execution after a certain time interval P_N. This interval is known as time period and it is the exact time elapsed between the two consecutive iterations of graph node. For example in Fig.2A P_{n1} defines the time period of node n_1.
- DMG is assigned to the scheduler in order to compute the schedule length (make span). Scheduling of DMG is non-preemptive. The cores process only one query at a time [13].

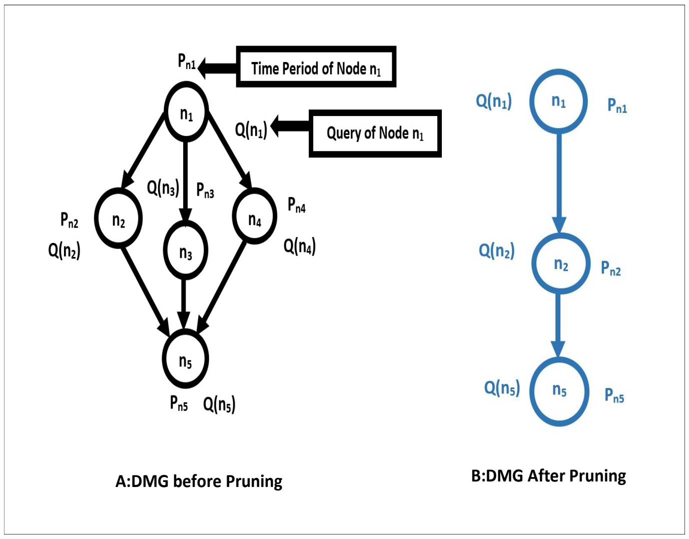

Fig. 2: Representation of DMG Before and After Pruning

TABLE I: Graph Pruning Constraints

Constraints	Notation
C_1	$Level(n_1) = level(n_2)$
C_2	$P(n_1) = P(n_2)$
C_3	$NotDel(\forall C(n_i) \in Level(DMG))$
C_4	$SO(Q(n_1)) = SO(Q(n_2))$
C_5	$DT(Q(n_1)) = DT(Q(n_2))$

- A bigger size of the DMG means that there are more diagnostic queries to process and a DMG will have a bigger make span. The optimization of the DMG should decrease the overall make span of the DMG in a manner that it does not violate the system timing constraints.

IV. Graph Pruning Algorithm

The basic objective of graph pruning is to optimize the graph by deleting different nodes based on constraints. The constraints direct the pruning of the DMG in a direction of minimizing the make span. The node deletion is not simpler like simple pruning as each node has a certain fault diagnostic query associated with it. Whenever a node of the DMG is deleted, its query should be transfered and merged with its neighboring node. If the constraints related to the query merging and node deletion are not satisfied then the node pruning of the DMG is not possible. These constraints are described in Table I.

For describing these constraints Fig.2 is used.

- C_1: It is the first constraint which needs to be satisfied as a first part of a DMG pruning algorithm. For deleting a node of the DMG and merging its query with its neighborhood node, it is important that both nodes should be present at the same depth level. Depth level of DMG is considered to be the distance from the root node to the specified node. If the nodes n_3 and n_4 are deleted in Fig.2A then the queries of nodes n_3 and n_4 can only be merged with the query of node n_2 because all of the three nodes n_2, n_3 and n_4 have the same depth level within the DMG. It is not possible that node n_4 is deleted and its query is merged with the node n_5.

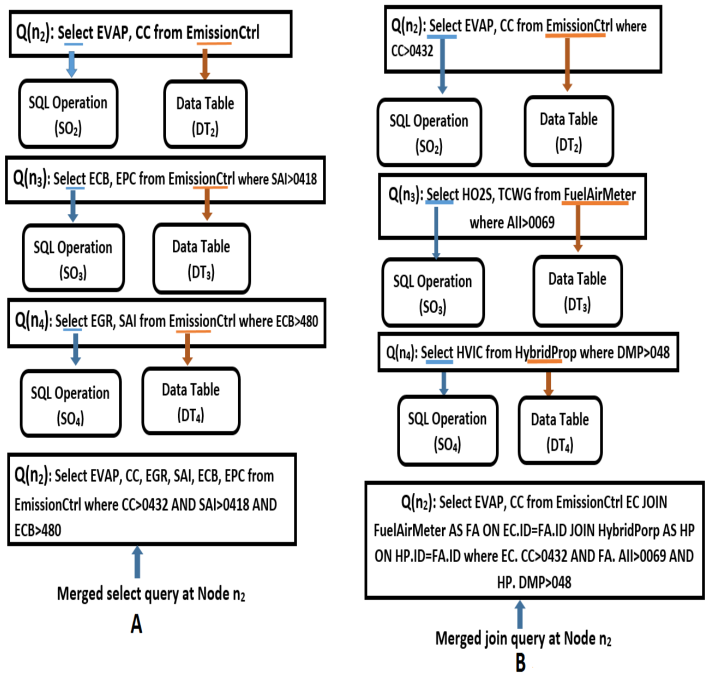
Q(n₂): Select EVAP, CC from EmissionCtrl
SQL Operation (SO₂)
Data Table (DT₂)
Q(n₃): Select ECB, EPC from EmissionCtrl where SAI>0418
SQL Operation (SO₃)
Data Table (DT₃)
Q(n₄): Select EGR, SAI from EmissionCtrl where ECB>480
SQL Operation (SO₄)
Data Table (DT₄)
Q(n₂): Select EVAP, CC, EGR, SAI, ECB, EPC from EmissionCtrl where CC>0432 AND SAI>0418 AND ECB>480
Merged select query at Node n₂
A
Q(n₂): Select EVAP, CC from EmissionCtrl where CC>0432
SQL Operation (SO₂)
Data Table (DT₂)
Q(n₃): Select HO2S, TCWG from FuelAirMeter where AII>0069
SQL Operation (SO₃)
Data Table (DT₃)
Q(n₄): Select HVIC from HybridProp where DMP>048
SQL Operation (SO₄)
Data Table (DT₄)
Q(n₂): Select EVAP, CC from EmissionCtrl EC JOIN FuelAirMeter AS FA ON EC.ID=FA.ID JOIN HybridPorp AS HP ON HP.ID=FA.ID where EC. CC>0432 AND FA. AII>0069 AND HP. DMP>048
Merged join query at Node n₂
B

Fig. 3: Query Merging: Case 1 and Case 2

- C_2: Both the node which is deleted and the one which is accepting the query of the deleted node, should have the same parent. Nodes n_2, n_3 and n_4 have the same parent node n_1.
- C_3: Not all children nodes belonging to one parent can be deleted. For example in Fig.2A if nodes n_3 and n_4 are deleted then n_2 cannot be deleted.
- C_4: Constraint C_4 and C_5 are based on the queries present within each DMG node. Each query $(Q(n_i))$ 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 query is split into parts, each query clause is matched with the query of its neighborhood node with which it has to be merged. $SO(Q(n_i))$ represents the SQL operation present in each query node of the DMG. For merging the queries the "select" operation is always considered. If the two merging queries have two different SQL operations namely "select" and "insert" then they cannot be merged. For example in Fig.2A if the $SO(Q(n_3)) = SO(Q(n_4))$ then queries of node n_3 and n_4 can be merged.
- C_5: After the constraint C_4 is satisfied $DT(Q(n_i))$ represents the database tables present in each query $Q(n_i)$. Two cases are considered on the basis of database tables within the merging queries.

A. Case 1

In the first case we consider that $DT(Q(n_1)) = DT(Q(n_2))$. It means that two queries have the same database tables from which they are fetching their tuples. If $DT(Q(n_2)) =$

$DT(Q(n_3)) = DT(Q(n_4))$ then at this point the process of deleting the nodes n_3 and n_4 and merging their queries with the node n_2 can be performed. Fig.3A shows the process for Case 1.

B. Case 2

In this case we consider that $DT(Q(n_1)) \neq DT(Q(n_2))$. The merged queries do not belongs to the same database table. These queries are merged together by using the join operation as the data table in each query is different. Fig.3B shows the process for Case 2.

The graph pruning algorithm works in cycles and prunes the DMG until the set of constraints in Table I is found and satisfied. The output of the pruning algorithm is the DMG shown in the Fig.2B.

Algorithm 1 Graph Pruning

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

C. Query Optimization

After the graph pruning next step is to perform query optimization.

1) The best access method is determine for each database table in a query. Access methods can be Index Scan (IS) or Table Scan (TS). The access method with minimum number of processing tuples is considered for optimization of queries.
2) The best join method for each join query is also determine. Join methods can be Hash Join (HJ), Merge Join (MJ) or Equi-join. The method with minimum processing tuples is always considered.
3) For queries having more than two join tables, always the best join order with minimum selectivity ratio is selected. First of all the different join orders are generated by using the left bushy tree [14]. The left bushy tree minimizes the number of join orders present in the search space. After that for each join order the selectivity ratio is calculated [15]. The selectivity factor can be calculated by equation (5).

$$S = \frac{|R_1 \bowtie R_2|}{|R_1|.|R_2|} \qquad (5)$$

4) On the basis of the above optimization techniques, the best QEP is generated for each query within the DMG. The best QEP for Fig.3B query $Q(n_2)$ is shown in Fig.4B. The cost for each QEP is calculated by estimating the total execution time it may take.

D. Scheduler

In our proposed system the last important block is the scheduler. The DMG along with the WCET of each query within the node is sent as an input to the scheduler. The scheduler calculates the make span of the DMG by assigning each node of the DMG to processors depending on the availability of processing resources. We are using scheduler designed in our previous work . Description about the scheduler is out of scope of this paper. More information about the scheduler can be studied from [16].

V. Calculation of Worst Case Execution Time

The worst case execution time of queries $WCET(Q(n_i))$, is an important input for real time systems [17]. The WCET is an input for the scheduler. The WCET is the maximum time a query may take during its worst case. For calculating the WCET of each query, different QEPs are considered and one with the maximum time to process the tuples is selected. Worst case QEP for query $Q(n_2)$ in Fig.3B is shown in Fig.4A.

VI. Illustrative Example

A. Queries in DMG Before Optimization

The data from sensors is received by the root Node A. The queries that store the sensor data into database table are actually insert queries. The three insert queries are executed at the Node A according to its time period 3. The example queries are synthetic. The fault codes terminologies used in these queries are extracted from [18] given in Table II .

Our example DMG is shown in Fig.5A.

- QA_1: Insert into EmissionCtrl (ID, CC, EGR, SAI, EVAP, ECB, EPC) VALUES (1, 0425, 0400, 0410, 0440, 0480,0496)

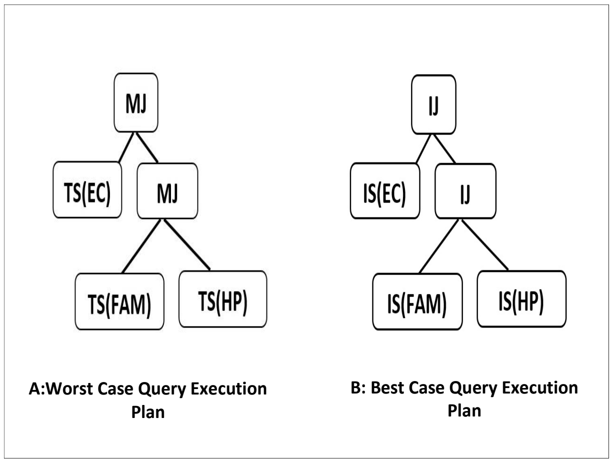
MJ
TS(EC)
MJ
TS(FAM)
TS(HP)
IJ
IS(EC)
IJ
IS(FAM)
IS(HP)
A:Worst Case Query Execution Plan
B: Best Case Query Execution Plan

Fig. 4: Worst Case, best Case QEP for Fig.3B query $Q(n_2)$

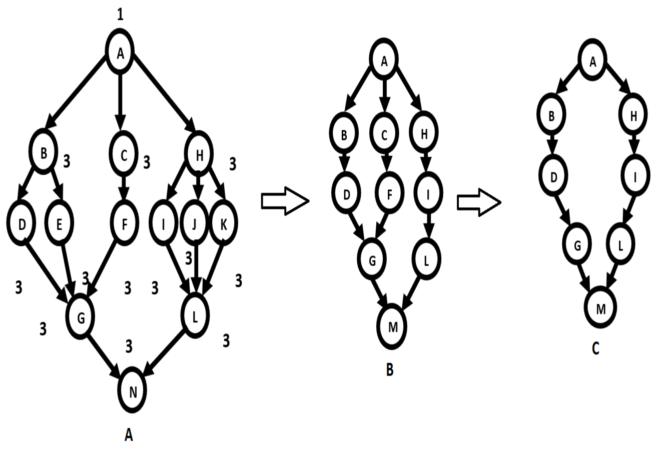

Fig. 5: Example DMG

TABLE II: Sensors Used in Example Queries

Constraints	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

- QA_2: Insert into FuelAirMeter (ID, HO2S, TCWG, CAMSHAFTP, AII, OAT, FPR, 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 queries in the parent Node A ends, 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 insert queries at Node A ends, it 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 are as follows:

- Node D, E: These two nodes are child node 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 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 $QG.QBCC > 04014$

1) Graph Optimization Cycle 1: This section will describe that how our graph optimization pruning algorithm works. As described in Section IV the graph pruning follows the steps describe in Section IV.

- Step 1: The graph pruning algorithm checks the maximum number of nodes at each level according to C_1. The maximum number of nodes are present at the second level of our DMG.
- Step 2: At this step the algorithm checks that parents of all neighboring nodes at selected level 2 should be the same. At level 2 Nodes D and E can merge 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, all child nodes cannot be deleted. So nodes E, J, K can be deleted only.
- Step 4: According to C_4, the similarity between the SQL operations of prospective merging nodes are checked. Queries in Q_D and Q_E are checked for similar "select" operation. According to queries mentioned in subsection A, Q_D and Q_E have the same SQL operations "select" so their queries can be merge. Similarly queries Q_I, Q_J and Q_K have the same operation "select".
- Step 5: This is the Case 1 of graph pruning as described in the Section IV. According to the operation C_5 and case 1 of pruning, the data tables of merging queries should be the same. According to the queries shown in subsection A, queries Q_D and Q_E have same the data table which is *QB*. Queries Q_I, Q_J and Q_K have same data table *QH*. After the queries are merged the resultant queries are as follow.
- 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 $MVAF <> 0299$. So the new graph generated after optimization cycle 1 is shown in Fig.5B. All the other queries at each node remain the same.

2) Graph Optimization Cycle 2: After the first step of optimization is completed, the algorithm will again check the new generated graph in order to determine whether additional pruning is require or not. If the constraints in Table I are satisfied then the graph pruning algorithm again prunes the graph by using all the previous steps. There are two special cases of pruning which we consider at this step.

- Special Case 1: In this case Node H is not merged with Node B and C. For merging the left cluster of nodes are considered because of the common child Node G. To keep the data transference accurate the child Nodes G and L are considered and Nodes B, C and H cannot be merged. Now Fig.5B is considered.
- Special Case 2: The algorithm checks for operation C1. The maximum number of nodes are present at both the first and second level of the graph. So the algorithm takes the smaller level first. It is level 1 and Nodes B and C are considered. After merging Node B and Node 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 is stored into new table *QB*).

Now the algorithm goes to the second level of graph, where it has found the maximum number of nodes and merges them together again. 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 then the child also has to merge itself with its neighborhood node and with the matching query. The data tuples required by Node F are from its parent Node C. These tuples are also transfered to the Node B. Queries 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 the Fig.5C.

B. Final Execution of Queries in DMG

The final queries present in the graph shown in Fig.5C are described in this section. Before the execution of DMG, for each query the most optimized QEP with minimum cost is generated as describe in Section IV. The worst case QEP is also generated as described in Section V. The final queries at each level of DMG after pruning are as follows.

1) Level 1: Q_B: Select EVAP, CC, EGR, SAI, ECB, EPC from EmissionCtrl where $CC > 0432$ AND $SAI > 0418$. (Results of Q_B query is stored into new table *QB*)
Q_H: Select * from FuelAirMeter where $HO2S > 0030$ AND $TCWG > 0000$. (Results of Q_H query is stored into new table *QH*).

2) 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 query is stored into 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 query is stored into new table *QI*).

3) Level 3: Earlier the query Q_G was join query but after merging of its parents G has no more joins.
Q_G: Select CC, SAI, ECB from QD where $ECB <> 0000$

TABLE III: Resultant Fault Values

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

TABLE IV: Data Statistics for make span of DMG before optimization (Secs)

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
Make Span		4.16

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

4) Level 4: Final query At Node M has two parents so it is getting join from two parent table 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 $QG.QBCC > 04014$.

Resultant fault values after the final query at node M are shown in Table III after the pruning and query optimization of the DMG is completed. The DMG with its worst case execution time is given as an input to the scheduler of the system. The scheduler calculates the make span of the DMG. Table IV and Table V shows the make span of our example DMG before and after optimization which is calculated by the scheduler.

VII. Experimental Evaluation

In order to test the applicability of our results, a homogeneous system with a total of six nodes is considered. This system has four processing units and two routers. A testing program is written in C and runs on an Ubuntu Linux machine. The database is created in PSQL. After every 15ms our database is populated with new data generated by sensors. Data tables required by each graph node are replicated to a particular processing unit at which they are executed.

TABLE V: Data Statistics for make span of DMG after optimization (Secs)

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
Make Span		2.04

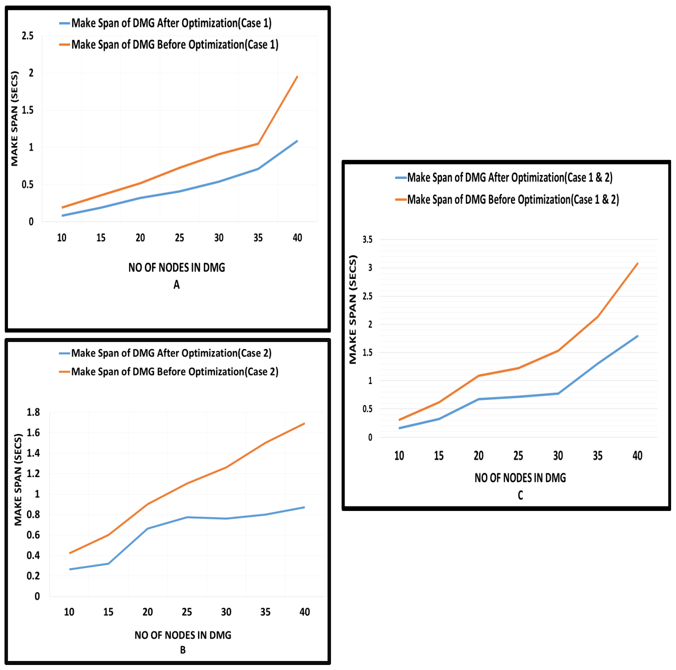

Fig. 6: Results

A. Types of Experimental Evaluation

For our experimental evaluation we have divided our DMG into three different categories.

1) Case 1: The graph pruning algorithm considers the DMG having the queries based on Case 1, described in Section IV. The merging nodes have different queries but they are accessing the same database table for their required set of data tuples.
2) Case 2: The graph pruning algorithm considers the DMG having the queries based on Case 2, describe in Section IV. In this case merging nodes have queries but they are accessing the different database tables for their required set of data tuples. Resultant merged queries in this case are join queries.
3) Case 3: In this case pruning algorithm considers the DMG having queries from both Case 1 and Case 2.

B. Complete Results

This section describes the complete results based on the graph pruning and best query execution plans applied to the DMG. The pruned and optimized DMG is given as an input to the scheduler with its WCET.

Fig.6A shows the results when our DMG is considering the Case 1 queries. All the nodes in which the queries accessing the data tuples from same tables are merged. In case of select queries the overall optimization applied is based on the selection of best access path. So the best QEP along with the pruned DMG minimized the overall make span of DMG up to half when our technique is implemented.

Fig.6B shows the results when our DMG is considering the Case 2 queries. Queries in this case are join queries which require more optimization, as the best join orders also have to select. In our case we consider the join for only three tables, in other words for each level of the graph, a maximum of three graph nodes can join. In the context of our results it is evident that when we have graphs with bigger size which require more pruning and more query optimization then our technique works better.

Fig.6C shows the results when our DMG is considering both cases. In this case our DMG has both types of queries. There are different numbers of select and join queries within the DMG. It is evident from the results that when we have more join queries and less select queries then our graph pruning and query optimization algorithm both works more efficiently.

VIII. CONCLUSIONS

This paper presents the graph pruning and query optimization technique to minimize the overall make span of a DMG. Different constraints are applied on the DMG and it is pruned. The queries of pruned nodes are merged with the neighborhood nodes of the DMG. After pruning the query optimization techniques are applied to the graph for calculating the best QEP. All these techniques minimize the make span of the DMG and enable it to complete within the time bound of the system. Minimizing the make span of DMG using genetic algorithm is considered to be the future work.

ACKNOWLEDGMENT

This project has received funding from the European Unions Horizon 2020 research and innovation program under grant agreement No 730830.

REFERENCES

[1] R. Isermann, FaultDiagnosis Systems: An Introduction from Fault Detection to Fault Tolerance, SpringerVerlag, 2005.
[2] 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, 2005.
[3] R.Obermaisser, R.Islam Sadat, and F.Weber, Active diagnosis in distributed embedded systems based on the time-triggered execution of semantic web queries. IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), pp. 222 - 229, 2014.
[4] G. Bauer .et. al, Time-Triggered Communication. Embedded Systems Series. CRC Press, Taylor and Francis Group, 2011.
[5] C. Bichot and P. Siarry, Eds.,Graph Partitioning. Hoboken, NJ, USA: Wiley, 2011.
[6] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz,Recent advances in graph partitioning, in Algorithm Eng. Sel. Results and Surveys, pp. 117158, 2016
[7] J. Zhang, F.Dong, D. Shen, J. Jin, J. Luo Superblock: An Application-aware Dynamic Partition Strategy for Large-Scale Graph,Third International Conference on Advanced Cloud and Big Data, Yangzhou, china, 2015
[8] Andersen, Reid, Fan Chung, and Kevin Lang, Local Graph Partitioning using PageRank Vectors. San Diego, California: University of California, 2006.
[9] Han M, Daudjee K, Ammar K, et al. An experimental comparison of pregel-like graph processing systems. Proceedings of the VLDB Endowment, 2014, 7(12), pp. 1047-1058.
[10] Y.Wang, Y. Mao, Z.Xu and P.Ping, Graph Partition Approach Based on the Cauchy Mutation and Inertia Weight , 14th Web Information Systems and Applications Conference, China, 2017.
[11] U. Benlic and J. HaoA, Multilevel Memetic Approach for Improving Graph k-Partitions,IEEE Transactions on Evolutionary Computation, 15(5), 2011
[12] H.Meyerhenke, P.Sanders, and C. Schulz, Parallel Graph Partitioning for Complex Networks, IEEE Transactions on Parallel and Distributed Systems, (28)9,2017
[13] B. Suger, G. D. Tipaldi, L. Spinello, and W. Burgard, An approach to solving large-scale slam problems with a small memory footprint, in 2014 IEEE International Conference on Robotics and Automation (ICRA). Hong Kong, pp. 3632 3637.
[14] Y.E.Ioannidis, Y.Kang, Left-deep vs Bushy Trees: An Analysis of Strategy Spaces and Its Implications for Query Optimization, ACM SIGMOD Conference on Management of Data, Denver, USA, 1991,
[15] B. Plale; K. Schwan, Optimizations Enabled by a Relational Data Model View to Querying Data Streams, 15th International Symposium on Parallel and Distributed Processing, USA, 2001
[16] S.Amin and R.Obermaisser, Time-Triggered Scheduling of Query Executions for Active Diagnosis in Distributed Real-Time Systems, 22nd International Conference on Emerging Technologies And Factory Automation, Cyprus, 2017
[17] A. Munnich and G. Farber,Calculating worst-case execution times of transactions in databases for event-driven, hard real-time embedded systems", International Symposium on DEA, 2000.
[18] Fault Codes, http://www.troublecodes.net/pcodes/