Chapter 4

Materialized View Selection by Evolutionary Algorithm for Big Data Query Processing

4.1 Introduction

Data warehouses traditionally support structured data because they are tied with operational or transactional systems for analytical processing by financial analysts and business lines for making decision on business strategies. However, with the advent of Big data, the conventional data warehouse concept is now changing because organizations are going for expanding and modifying the data warehouse for relevance in Big data environment of huge volume of semi structured or *key-value paired* [76] data in highly distributed computational framework.

A computing paradigm called MapReduce have been introduced in Big data [16]. In MapReduce model of Big data systems, computation tasks are broken up into units that can be distributed around a cluster of commodity hardware and server class hardware for providing cost-effective processing with scalability. The MapReduce system was designed and is best suited for handling big and semi structured data such that each split of the data for MapReduce computation is from a single big table or file instead of large number of small files. This is because in Distributed File System (DFS) based Big data processing such as *Hadoop Distributed File System* (HDFS), the block size is kept as at least 64MB (or multiple of that), as in this frame work of processing, a very large number of distributed commodity memory is available and smaller file size of less than the block size imposes unnecessary overhead. The technologies used in Big data warehousing organizes data into tables as a mean for providing structure to data stored in DFS and supporting row-level update like *delete, update* etc. on big column oriented table of key-value type storage. These technologies support *collection data type* columns STRUCT, MAP and ARRAY sacrificing *normal form* for higher processing throughput [18]. Again, as these technologies support collection data type,

therefore they are designed with very limited indexing capabilities. Hence, in Big data systems, for analytical processing queries, the use of indexes and keys in the relations are very limited. And therefore, common or shared sub-expression results of frequent queries may be utilized as materialized views without the need of designing a common optimized query execution plan based on indexes and relations for the considered set of queries.

4.1.1 Materialized views and materialized queries in Big data

Big data analysis by Distributed File System (DFS) is a cost-effective framework that binds very large data sets in a cluster of computers into a pool for distributed processing [77]. The imposed programming model termed as *MapReduce* breaks-up computation tasks into smaller jobs for distributing them around the data created by splitting large amount of data into a cluster of commodity computer hardware for distributed processing [17, 77]. The distributed file system (DFS) used in Big data by *Apache* [78] is termed as Hadoop Distributed File System (HDFS) [17, 79]. For Data Warehousing applications in HDFS, Hadoop [79] provides a technology called *Hive* and an SQL like language called *HiveQL* [80]. In Big data the total MapReduce cost against generating responses of a set of queries depends on MapReduce splits. MapReduce costs are thus involved in creation of temporary views while processing queries. To make query responses faster, if these temporary views are saved for future query processing, MapReduce cost is also to be incurred for updating the views. The problem of view selection for materializing is that - a set of views, generated while processing a set of queries are to be selected for materializing, so that if this set is materialized or saved, the total query MapReduce cost and MapReduce cost for maintaining the materialized views are minimum.

Julian Hyde in [19] proposes an extension to materialized views called *Discardable, In-Memory Materialized Query - DIMMQ for Hadoop*. DIMMQ proposes that the resultant data-set of some frequent queries reside in the memory of one or more nodes in the cluster. Discardable means that the system can remove the in-memory materialized queries when they are not used for a long time. Here it is proposed that some sub-queries may be saved in the memory of hardware cluster with mapping to their resultant data in disk. But even here the MapReduce overheads for job submission and job scheduling remains along with the maintenance cost for refreshing the mapping between in-memory queries and the disk data. Therefore whether it is materialized views or in-memory materialized queries, a sub-set from the candidate set of views or queries are to be selected for materializing such that all related costs are minimized.

4.1.2 View selection for materializing in Big data

In Big data framework, a query is executed by accessing data spread over a cluster of hardware storage or data-nodes as distributed file system (DFS) [79] by MapReduce jobs. Therefore the query processing cost is not just the cost of accessing rows of base tables stored in disk. The DFS overhead of distributing data into data-nodes, mapping and tracking of processing, and then reducing the results also are involved. As every complex query may have several sub-queries with multiple number of aggregation functions, therefore either these sub-queries may be materialized in memory of hardware cluster with mapping to their resultant data in disk or the intermediate result of sub-queries may be materialized in disk as materialized views. Thus by materializing these intermediate views, the MapReduce cost of repeated processing of these sub-queries can be avoided. But the DFS overhead cost for materializing these views and refreshing them periodically are still to be incurred. The materialization of temporary views also needs space in the hardware cluster. Therefore a system may be designed to recommend a set of intermediate views so that if they are materialized, the total query processing cost savings for the selected set of frequent queries is maximized with minimized materialized view refreshing cost and space cost. Therefore the materialized view selection problem is defined as an optimization problem.

If, the number of queries considered as frequent queries increases, then the number of candidate views or sub-queries for materializing also increases. Different query processing costs and other associated costs for different combination of views are independent of the total number of views selected. Thus the solution space increases exponentially with increased number of queries and underlying views considered.

4.1.3 Contribution

In this chapter, design of a system is proposed for finding solution set of views for materializing to optimize query processing cost, materialized view maintenance cost, and storage for materialized views from views generated while processing a set of queries on Big data warehousing. The problem is defined as a multi-objective optimization problem for finding non-dominated solution set of views using Multi-objective Differential Evolution algorithm and Non-dominated Sorting Genetic Algorithm-NSGA-II [32]. Here, *forma analysis* based multi-objective DE for binary encoded data, termed as MODE-BE, presented in Chapter 3 is modified and implemented for selecting views for materializing in Big data framework based data warehouse by promoting diversity in solution vector space. In NSGA-II the diversity of large number of solutions are promoted by computing crowding distances between solutions in objective function value space. The NSGA-II is also implemented on this problem to analyze performances between NSGA-II based systems and MODE-BE based recommendation systems for materialized view selection in Big data management.

The problem of selecting views for materializing in Big data warehousing is defined in Section 4.2. In Section 4.3, materialized view selection in Big data framework has been defined as a multi-objective optimization problem. An improved version of Multi-objective Differential Evolution algorithm with Binary Encoded solution representation for materialized view selection, MODE-BE, and implementation of Non-dominated sorting Genetic Algorithm, NSGA-II, in materialized view selection for Big Data warehousing are presented in Section 4.4. In Section 4.5, the experimentation process of implementation of algorithms and the test data used in the experimentation are presented along with an analysis on obtained results. Finally in Section 4.6 concluding discussion and perspectives are presented.

4.2 The Problem of Selecting Views for Materializing in Big data Framework

To make query response faster, a set of views are to be selected for materializing to minimize total query response cost of a set of frequent data warehouse queries with optimum maintenance cost or updating cost of the materialized views. Hive uses the advantage of Big data framework's scale out and robust capabilities for data storage and processing on large number of commodity hardware. HiveQL enables to do ad-hoc query processing, summarization and data analysis on massive data easily [80]. The DFS and MapReduce paradigm are used for working with massive data for storage and analysis at Internet scale which is otherwise unmanageable by conventional data processing with database management system. HiveQL [80] query processing on Hadoop version 1 often had to submit number of MapReduce jobs to complete a query processing. With Hadoop-2 and *Tez* platform the cost of job submission and scheduling is minimized by removing the restriction that the jobs are to be done only by Map and Reduce for all kind of processing [20]. But in general, for Big data, processing standard is by MapReduce [79]. Though Map tasks write intermediate output to the local disks, input to a single Reduce task is normally the output from all Map tasks. The Map outputs are transferred across the network to the node running Reduce tasks and the merged output is to be passed to the user-defined Reduce functions. Thus the intermediate MapReduce result sets are needed to be stored in DFS and thereby the MapReduce jobs in the system degrades the system performance. Also submitting jobs and scheduling them across the DFS adds extra costs [20].

4.2.1 The cost model and problem definition

The cost model to be used for handling materialized view selection problem for Big data system is different from cost models used in approaches used for conventional Client-Server architecture with RDBMS based data warehousing. The main reason behind this is that, in conventional RDBMS based system the data access pattern is mainly dominated by "seeks" and "seek time", whereas in Big data DFS or in

similar distributed framework, the data access pattern is mainly dominated by data transfer rate and MapReduce costs. The MapReduce paradigm is designed to analyze massive amount of data in batch fashion unlike the traditional RDBMS where data-set has been indexed to deliver low-latency seek and update time [77]. As for increased size of data, a bigger sized commodity hardware cluster may be used, therefore the performance of MapReduce functions are independent of size of the data or rows to be accessed.

The MapReduce overheads are composed of data transfer cost of transferring data into number of data nodes across the DFS cluster, running *job trackers* and *task trackers*, creation of Mapppers and Reducers and substantial overheads in job submission and scheduling. In Big data management DFS, block size and split size are fixed. Therefore, in Big data/MapReduce framework, a small number of large files are better than large number of small files [17]. This means that in case of Big data based data warehousing smaller number of bigger views are to be preferred for materializing. This criterion is not applicable in case of traditional RDBMS based data warehouse's materialized views. The different costs and benefits that are to be optimized for materializing views to enhance query processing in Big data warehousing are formally defined below.

4.2.1.1 Query processing cost

The total query processing cost of a set of frequent queries may be considered as the total MapReduce overheads for executing the set of queries. If the results of some sub-queries and aggregate functions used in these queries are materialized or saved, then in subsequent execution of the queries, MapReduce overheads of executing these sub-queries or views are saved. If a set of sub-queries of a set of frequent queries are processed and composed as views and materialized in Big data DFS, the query processing cost of the set of considered queries may be defined as Definition 19.

Definition 19. *For a set of n number of frequent queries $Q = \{q_1, q_2, \cdots, q_n\}$ on a data warehouse, where V is the set of m intermediate views generated by Q, if $V' \subseteq V$ is the set of views $V' = \{v_1, v_2, \cdots, v_p\}$ that are materialized, the total Big data query processing cost can be defined by the following expression.*

$$C^Q_{V'} = C^Q_{\varnothing} - \sum_{i=1}^{p} M_{v_i} \tag{4.1}$$

where $C^Q_{\varnothing} = \sum_{i=1}^{n} M_{q_i}$ is the total query processing MapReduce cost of Q without materializing any view, and $\sum_{i=1}^{p} M_{v_i}$ is the MapReduce cost of processing V'.

4.2.1.2 Materialized view maintenance cost

In analytical processing on DFS based Big data warehouse, there are generally very few occurrences of updating operations. But whenever there is a change in

the base data, the materialized views are to be updated. In case of in-memory query materialization, frequent refreshment is needed as in this case infrequent queries are to be discarded after each fixed period of time [19]. Materialized view maintenance means re-processing the aggregate functions and/or corresponding sub-queries and then updating the views in disks or solid state drives[1]. Thus there will be another set of DFS overheads. The materialized view maintenance cost may be defined as follows.

Definition 20. *For a set of materialized views* $V' = \{v_1, v_2, \cdots, v_p\}$ *for processing a set of queries* Q*, the materialized view maintenance cost may be expressed as*

$$U(V') = \sum_{i=1}^{p} U_{v_i} \tag{4.2}$$

where U_{v_i}*,* $i = 1, 2 \cdots, p$ *, are the maintenance MapReduce overheads for the set of materialized views* $v_i \in V'$*,* $i = 1, 2 \cdots p$*.*

4.2.1.3 Number of views to be materialized and storage space requirements

The storage space requirements for p number of materialized views V' ($|V'| = p$), can be defined as Definition 21 below. In Big data systems, smaller number of bigger tables are preferred (as discussed in Section 4.2.1). That is, $|V'| = p$ is to be minimized with maximized size of the tables.

Definition 21. *If* A_{v_i} *is the storage space required by* i*th materialized view, then the total space required for materializing* p *number of views is*

$$A_{V'} = \sum_{i=1}^{p} A_{v_i} \tag{4.3}$$

4.2.1.4 The materialized view selection problem

Considering the definitions 19, 20 and 21, the view selection for materializing in Big data based data warehousing can be stated as Definition 22 below.

Definition 22. *The view selection for materializing in Big data framework data warehousing for a given set of* n *frequent data warehouse queries* $Q = \{q_1, q_2, \cdots, q_n\}$*, where* V *is a set of* m *views generated while processing* Q*, a set of views* $V' = \{v_1, v_2, \cdots, v_p\}$*,* $V' \subseteq V$ *i.e.* $p \leq m$*, is to be selected such that it minimizes*

1. $C_{V'}^{Q}$ *defined by Equation 4.1,*

[1]Here in addition to *select, join, project* and aggregation functions, MapReduce overhead due to *update* is incurred.

2. *$U(V')$ defined by Equation 4.2 and*

3. *$|V'| = p$ with constraint on minimum value of $A_{V'}$ defined by Equation 4.3.*

In next section we define materialized view selection as a multi-objective optimization problem and present a discussion on applying Multi-Objective Evolutionary Algorithm (MOEA) for solving this problem.

4.3 View Selection in Big data Systems as Multi-Objective Optimization Problem

From the above definitions 19,20, 21 and problem statement in Section 4.2.1.4, for a given set of views, say V, the view selection problem is to find the set V', $V' \subseteq V$, to minimize -

$$\mathbf{Y} = \mathbf{F}(V') \equiv (C^Q_{V'}, U(V'), |V'|) \tag{4.4}$$

such that the constraint on amount of space, $A_{V'}$, for materializing V' is as specified by user.

4.3.1 Simple problem representation

Deb et al. in [60] suggest few important features that must be present in an multi-objective optimization problem for solving by randomized and evolutionary algorithm. According to [60], very importantly the problem should be easy to construct with known dimensions. In our problem definition it is assumed that a set of frequently processed queries are known and thereby the frequent temporary views or sub-queries and aggregate functions triggered on the data warehouse can be derived or known. In our definition this known set of views are defined as V, where the cardinality of V is m, i.e $|V| = m$, and the cardinality of selected views for materializing V', $|V'| = p$. As $p \leq m$, a solution vector may be defined as a string of bits of length m where each of the m dimension may be represented as a decision variable that may be either selected or not selected for materializing.

In our representation of the problem and solution, we have labeled each of the candidate views with a serial number starting from 1 to m for m dimensions of each solution vector. In solution string, the first bit represents the candidate view labeled as the first view, the second bit represents the view labeled as second view and so on. If a view is not selected then the corresponding bit i.e., the corresponding dimension in the candidate solution vector is set as 0 and otherwise, if the view is selected for materializing, its corresponding bit is set as 1.

For two solution strings, say S_0 and S_1 of length m, if $C^Q_{S_0}$ and $C^Q_{S_1}$ are the total query processing costs for a set of frequent query Q having m num-

ber of candidate views for materializing, $U(S_0)$ and $U(S_1)$ are the corresponding maintenance cost of the views if materialized, and if $num(S_0)$ and $num(S_1)$ are number of views selected in solution S_0 and S_1, then *iff* $C^Q_{S_0} \leq C^Q_{S_1}$ *and* $U(S_0) \leq U(S_1)$ *and* $num(S_0) \leq num(S_1)$, then *if* $C^Q_{S_0} < C^Q_{S_1}$ *or* $U(S_0) < U(S_1)$ *or* $num(S_0) < num(S_1)$, then the solution S_0 dominates solution S_1 which is expressed as $S_0 \prec S_1$. If S_0 does not dominate S_1 and S_1 also does not dominate S_0, expressed as $S_0 \not\prec S_1$ and $S_1 \not\prec S_0$, then S_0 and S_1 are two non-dominating solutions of the problem.

Definition 23. *The materialized view selection problem is the problem of finding the set of non-dominating solutions which is an approximation to the true Pareto front of the problem defined by Equation 4.4.*

4.3.2 Scalability

In [60], it has been suggested that the test problem for applying multi-objective optimization problem should be scalable. In our representation of the problem, the solution vectors are of dimension m, where m is the total number of candidate views. As each decision variable is expressed as a single dimension of a solution vector, the solution vector representation is linearly scalable with number of dimensions i.e., value of the variable m. For m number of decision variables of a solution vector, the size of the solution vector space will be 2^m. Thus with increasing dimension in decision vector space, the solution vector space increases exponentially. Due to this, stochastic, randomized or evolutionary algorithms are suitable for handling this problem.

4.3.3 Well defined objectives

For defining a problem as multi-objective optimization problem and solving it by evolutionary or randomized algorithm, most importantly the objectives should be distinct and well defined. In our problem definitions and by Equation 4.4, three objectives are clearly defined. With these three objectives a clearly visible Pareto-front or Pareto-optimal surface may be plotted for getting a clear idea of performance by a multi-objective optimization technique applied on this problem.

In Equation 4.1 and Equation 4.2, M_{v_i} and U_{v_i} for ith view v_i and M_{q_i} for ith query are independent variables. $|V'|$ cannot determine $C^Q_{V'}$ and $U(V')$, and $C^Q_{V'}$ cannot determine $|V'|$ and $U(V')$. Similarly $U(V')$ cannot determine $|V'|$ and $C^Q_{V'}$. Therefore, multi-objective optimization can be designed to introduce controllable hindrance to getting trapped in local optimum.

4.4 Multi-Objective Evolutionary Algorithm for View Selection to Materialize in Big data

It has been observed from our discussion presented in Chapter 3 that multi-objective Evolutionary Algorithms (MOEAs) are suitable for applying in materialized view selection problem. In solution representation for handling this problem, solution vectors have been defined as a string of bits. The single objective DE for binary encoded data as presented by Gong et al. in [31] has been customized and implemented for multi-objective optimization to handle materialized view selection problem for conventional data warehousing in Chapter 3. In basic MODE-BE, as implemented earlier in Chapter 3 for materialized view selection in conventional data warehousing, it has been observed that when the initial population, NP, is very large, there arises a memory issue because - before *mutation, cross-over* and *selection* are performed for all the NP solution vectors, the number of offspring solution vectors some times becomes too large to be handled. In basic MODE-BE, to control the population size in intermediate generations, the most elite solutions that maintain diversity in the population are retained discarding the other solution vector after the mutation, cross-over and selection are done for all NP solution vectors. In this improved version of MODE-BE for materialized view selection in Big data management framework, a threshold value is used to determine the maximum size of the population in intermediate generations such that whenever the offspring population size of a generation after mutation, cross-over and selection becomes more than this threshold value, the most elite solutions in terms of their Pareto rank and diversity measure in the population are retained for next generation discarding the rest of the offspring population generated. Selecting appropriate set of views for materializing out of a large number of solutions in the *first Pareto front* which may be positioned very closely in objective function space, is another issue in case of very large value of NP. Therefore in the proposed version of MODE-BE, a filtering criterion on maximum dissimilarity values among solutions is proposed based on the *Three Sigma Rule* [81]. The NSGA-II proposed by Deb et al. in [32] also has been implemented for comparative performance analysis between MODE-BE and NSGA-II in selecting views.

4.4.1 Multi-objective DE with binary encoded solutions for Big data view selection

For selecting materialized views for conventional data warehousing, the mutant vectors in multi-objective DE with binary encoded data generated by *forma basis* as discussed in [66,67] has been used in Chapter 3. In this work forma basis based multi-objective DE for binary encoded data (MODE-BE) has been used to design Algorithm 6 for selecting views to materialize in Big data framework based data warehousing.

In *DE/rand/1/bin* version of DE, the mutant vector for next generation

Algorithm 6: View selection for materializing by Multi-objective Differential Evolution using Binary Encoded Data in Big data based data warehouse.

Require: NP, g_{max}, F, CR, D, $C^Q_\varnothing$, $M_{v_{i=1,\cdots,m}}$, $U_{v_{i=1,\cdots,m}}$, $A_{v_{i=1,\cdots,m}}$, Γ , MinSpace

Ensure: A set of non-dominated solutions.

Generate NP random vectors $x_1, x_2, \cdots, x_{NP}$ of dimension D that satisfy the MinSpace constraint
$N \leftarrow NP$
$g \leftarrow 1$
repeat
  for $i = 1$ to N **do**
    select x_i and $x_{r_1}, x_{r_2}, x_{r_3}$, such that $x_i \neq x_{r_1} \neq x_{r_2} \neq x_{r_3}$
    for $j = 1$ to D **do**
      $v_{j,i} \leftarrow D_{\Psi_j}(x_{r_1}, round(F.D_{\Psi_j}(x_{r_2}, x_{r_3})))$
where $round(F.D_{\Psi_j}(x_{r_2,g}, x_{r_3,g})) =$

$$\begin{cases} 1, & \text{if random}[0,1] < F \wedge (D_{\Psi_j}(x_{r_2,g}, x_{r_3,g}) = 1) \\ 0, & \text{otherwise} \end{cases}$$

$$D_{\Psi_j}(x_{r_2}, x_{r_3}) = \begin{cases} 0, & \text{if } x_{j,r_2} = x_{j,r_3} \\ 1, & \text{otherwise} \end{cases}$$

      $j \leftarrow j + 1$
    end for
    $rand_i = random[1, D]$
    for $j = 1$ to D **do**

$$u_{j,i} \leftarrow \begin{cases} v_{j,i}, & \text{if}(random[0,1] \leq CR) \text{ or } j = rand_i \\ x_{j,i}, & \text{otherwise} \end{cases}$$

      $j \leftarrow j + 1$
    end for
    if $space(u_i) \geq MinSpace$ **then**
      Evaluate query processing cost $C^Q_{x_i}$ and $C^Q_{u_i}$, materialized view maintenance cost $U(x_i)$, $U(u_i)$ and number of non-zero elements in u_i, x_i
      if $u_i \prec x_i$ **then**
        $x_i \leftarrow u_i$
      else
        if $x_i \not\prec u_i$ **then**
          $NP \leftarrow NP + 1$
          $x_{NP} \leftarrow u_i$
        end if
      end if
    end if
    if $NP \geq \Gamma N$ **then**
      $i \leftarrow N + 1$
    else
      $i \leftarrow i + 1$
    end if
  end for
  if $NP > N$ **then**

Algorithm 7: View selection for materializing by Multi-objective Differential Evolution using Binary Encoded Data in Big data based data warehouse- (continued from previous page).

Compute and assign Pareto rank to each of the population vectors $x_{i=1,\cdots NP}$
Compute maximum distance of each solution vector to other solution vectors of the population, Max_i
Sort vectors $x_1, x_2, \cdots, x_{NP}$ in ascending order of Pareto-rank and descending order of Max_i
$NP \leftarrow N$
end if
$g \leftarrow g + 1$
until $(g < g_{max})$
Remove all dominated solutions from the population list $x_1, x_2, \cdots, x_{NP}$
Return The final set of solution population

$g+1$ for each target vector $x_{i,g}$, $i = 1, 2, \cdots, NP$, is generated as equation 4.5.

$$v_{i,g+1} = x_{r_1,g} + F.(x_{r_2,g} - x_{r_3,g}) \tag{4.5}$$

where $r_1, r_2, r_3 \in \{1, 2, \cdots, NP\}$, $r_1 \neq r_2 \neq r_3$, F is a real constant factor $\in [0, 1]$ and $F > 0$. By using forma basis [66, 67], Gong et al. in [31] expressed mutant vector defined by Equation 4.5 for binary encoded solution vector as Equation 4.6.

$$v_{j,i,g+1} = D_{\Psi_j}(x_{r_1,g}, F.D_{\Psi_j}(x_{r_2,g}, x_{r_3,g})) \tag{4.6}$$

Where, $x_{r_2,g}$ and $x_{r_3,g}$ are considered as two strings of bits of length D and each jth dimension difference between $x_{r_2,g}$ and $x_{r_3,g}$, $D_{\Psi_j}(x_{r_2,g}, x_{r_3,g})$ is represented by using *formae basis* [31] Ψ_j as Equation 4.7.

$$D_{\Psi_j}(\mathbf{x}, \mathbf{y}) = \begin{cases} 0, & \text{if } x_j = y_j \\ 1, & \text{otherwise} \end{cases} \tag{4.7}$$

To interpret the scaled difference $F.D_{\Psi_j}(x_{r_2,g}, x_{r_3,g})$ of jth dimension rounded to 1 or 0, Equation 4.8 is used.

$$F.D_{\Psi_j}(x_{r_2,g}, x_{r_3,g}) = \begin{cases} 1, & \text{if random}[0,1] < F \wedge (D_{\Psi_j}(x_{r_2,g}, x_{r_3,g}) = 1) \\ 0, & \text{otherwise} \end{cases} \tag{4.8}$$

In Algorithm 6, a set of initial solutions $x_1, x_2, \cdots, x_{NP}$ are generated for a given set of frequent queries Q that satisfy the space constraint. In each generation of a evolutionary process g, against each solution vector $x_i, i = 1, 2, \cdots, NP$, a mutant vector $v_{i,g+1}$ is generated as expressed by Equation 4.6 (and Equation 4.8). Then trial vector $u_{i,g+1}$ is formed by crossover. To adapt the problem of view selection to materialize in Big data, the query processing cost, the materialized view maintenance cost and the number of views in solution sets of the considered frequent queries Q, for each solution vector $x_{i,g}$ and trial vector $u_{i,g+1}$ are computed

by using equations (4.1),(4.2) and by counting number of selected views in $x_{i,g}$ and $u_{i,g+1}$. If $u_{i,g+1} \prec x_{i,g}$, then $x_{i,g+1}$ is set as $u_{i,g+1}$, else if $x_{i,g} \prec u_{i,g+1}$, then $u_{i,g+1}$ is discarded. Otherwise, in case $u_{i,g+1} \not\prec x_{i,g}$ and $x_{i,g} \not\prec u_{i,g+1}$, $u_{i,g+1}$ is appended to the population for next generation $g+1$. Thus the population may go on increasing. To control the population growth in each generation of DE, when the population size touches a limit, i.e when NP becomes ΓN, N being the initial population size (i.e initial NP), and Γ being a positive real constant, a technique is used to filter out NP elite solution population that maintains diversity in the solution population as discussed in Section 4.4.1.1. This evolutionary process is continued till it reaches a maximum number of generations specified, say g_{max}. The dominated solutions in the final population are then removed from the archive to return the non-dominated solutions of the problem.

4.4.1.1 Promoting elitism and diversity in solution population

For elitism and diversity in solution population, though the Pareto ranking has been used as suggested by Deb et al. in [32], the diversity of solutions are maintained in solution space unlike it is done by using Crowding distance in objective function space for NSGA-II. The diversity in solution space is preferred here for wide coverage of representations of views in the solution set which has already been discussed in Chapter 3. The Pareto ranking and diversity preservation used in this version of MODE-BE are as stated below.

Pareto ranking: To control the population size by keeping the diversity in solution population in the intermediate generations of DE in this approach, the diversity of solutions in solution space is promoted with necessary elitism. When the population size NP in a generation becomes ΓN, where N is the initial value of NP in that generation, the solutions of intermediate generations are ranked according to their Pareto dominance levels as discussed in [32]. The solution population is then sorted in ascending order of their Pareto ranks so that the most elite solutions from the list may be kept in the population for next generation. To ensure finding out the most elite NP solutions, Deb et al. [32] suggests $\Gamma = 2$.

Diversity in solution space: For each $i-th$ solution of the population, the maximum distance to other solution vectors in the population Max_i is measured. Since the solution vectors are represented as a string of bits and a particular bit as 1 or 0 does not mean any preference over each other, the *Simple Matching Co-efficient (SMC) distance* measure [72] is used for measuring distance between two solutions. The solution population sorted in ascending order of their Pareto ranks are then sorted again on descending order of their maximum distances to other solutions in the population. From the doubly sorted solution population, the top NP solutions are retained as next generation population. Here, to promote diversity of solutions, density in solution space is used instead of using crowding density measured by *crowding distance* of solutions in objective function value space.

4.4.1.2 Filtering Representative Solutions from Non-dominated Solutions Obtained

From large number of non-dominated solutions yielded by multi-objective optimization, finding significant representation set is useful for decision makers [82]. In selecting views for materialization in data warehouse, degree of diversity in solutions is important as solutions having larger representations from candidate views are preferable. Therefore, for filtering significant solutions from the obtained solutions, farthest distance approach may be incorporated. In our implementation, first the SMC based maximum distance from every solution vector S_i of n number of non dominated solutions to other solution vectors in the population, $Max_i, i = 1, 2, \cdots, n$, is computed. Then the mean μMax and standard deviation σMax of $Max_i, i = 1, 2, \cdots, n$, are computed. In the empirical sciences and in statistics the "three sigma rule of thumb" expresses a conventional heuristic that in a normal distribution, 68.27% of values lie within mean (μ) and one standard deviation(σ) i.e within $\mu \pm \sigma$, 95.45% values lie within mean and two standard deviation i.e within $\mu \pm 2\sigma$ and 99.73% values lie within mean and three standard deviation i.e within $\mu \pm 3\sigma$. Based on this "three sigma rule of thumb" popularly referred as "68-95-99.7 rule", for narrowing down the options of considering non dominated solutions, a solution S_i is discarded if $Max_i < (\mu Max + C.\sigma Max)$, where C is a real constant that may be specified as positive, negative or zero, depending on number of solutions we want to filter out from the population, preserving diversity in solution space.

4.4.2 Implementing NSGA-II for view selection

In [26], two implementations of multi-objective GA for materialized view selection have been presented. These two approaches used non-elitist multi-objective evolutionary algorithms for selecting views under storage space constraint. Deb et al. in [32] presented that elitism can speed-up the performances of the GA significantly and also can help retaining good solutions generated during intermediate generations. In multi-objective GAs for ensuring diversity in solution population, the concept of sharing parameter σ_{share} in objective space is used. But a parameter less diversity preservation mechanism is better. To address this issue NSGA-II was suggested [32].

To adapt NSGA-II for selecting views to materialize in Big data warehousing with distributed file system framework, I used the cost model and problem definition as discussed in Section 4.2 and 4.3.

Here, first a random solution population $\Psi_1, \Psi_2, \cdots, \Psi_{NP}$ are created using the same solution representation as used in MODE-BE. For generating ith random solution, initially all decision variables of Ψ_i are set as 0. A random integer in the range [0,D] is generated for deciding how many dimensions of Ψ_i is to be set as 1. This random number is the cardinality of the set of views selected V' in Ψ_i, expressed as $|V'| = p$ in Definition 22. Randomly these p number of

decision variables are set as 1 for the vector Ψ_i. As discussed in Section 4.2, in Big data framework based data warehousing (like Hive), smaller number of larger sized views are to be selected for materializing. Therefore, the solution vector Ψ_i is added to the list of initial population only if the total size of the set of p number of views of the set V' satisfies the minimum space criteria specified and Ψ_i is not already present in the solution population.

In subsequent generations, the usual binary tournament selection, recombination and mutation operators are applied to the NP solutions to create offspring. In each generation, against a selected solution from the NP solutions, one offspring is generated. For finding domination or non-domination between two solutions, say Ψ_i and Ψ_j, where V_i' is the set of non-zero dimensions of Ψ_i and V_j' is the set of non-zero dimensions of Ψ_j, the three objective functions (1) the query processing costs $C^Q_{V_i'}$, $C^Q_{V_j'}$, (2) maintenance costs $U(V_i')$, and $U(V_j')$ and (3) $|V_i'|$, $|V_j'|$ are evaluated. If the generated offspring dominates the selected solution vector, then the new offspring replaces the selected vector. Otherwise , if the newly generated offspring does not dominate the solution vector and if the selected solution also does not dominate the offspring, the offspring vector is added to the population. Thus a new offspring population of size N is created. Whenever N becomes $2NP$, the solution population in the list are ranked in their non-domination levels. The ranked solution population are sorted in ascending order of their non-domination ranks. The crowding distance among the solutions are then computed in objective function space and sorted in descending order of their crowding distances. The solutions are sorted in ascending order of ranks for providing higher priority for keeping the solutions of lower domination ranks in next generation, so that the most elite solutions are retained in subsequent generations. The solution population are sorted in descending order of objective function based crowding distances to preserve diversity among solution population in each generation.

4.4.2.1 Run time complexity of NSGA-II

The basic operations of NSGA-II based application's worst case complexities as presented in [32] are - (1) for non-dominated sorting is $O(M(2N)^2)$, (2) for crowding distance assignments is $O(M(2N)log(2N))$ and (3) for sorting on crowding distances is $O(2Nlog(2N))$. Here, M is the number of objectives and N is the number of solution population. Thus the over all complexity is dominated by $O(M(2N)^2)$. As our problem is defined with 3 objectives therefore it becomes $O(3(2N)^2)$. Thus the overall complexity is $O(N^2)$.

4.5 Experimentation and Observations

For experimental analysis, Multi-Objective DE for Binary Encoded solutions, MODE-BE, and NSGA-II as a recommending system, taking input from log-files

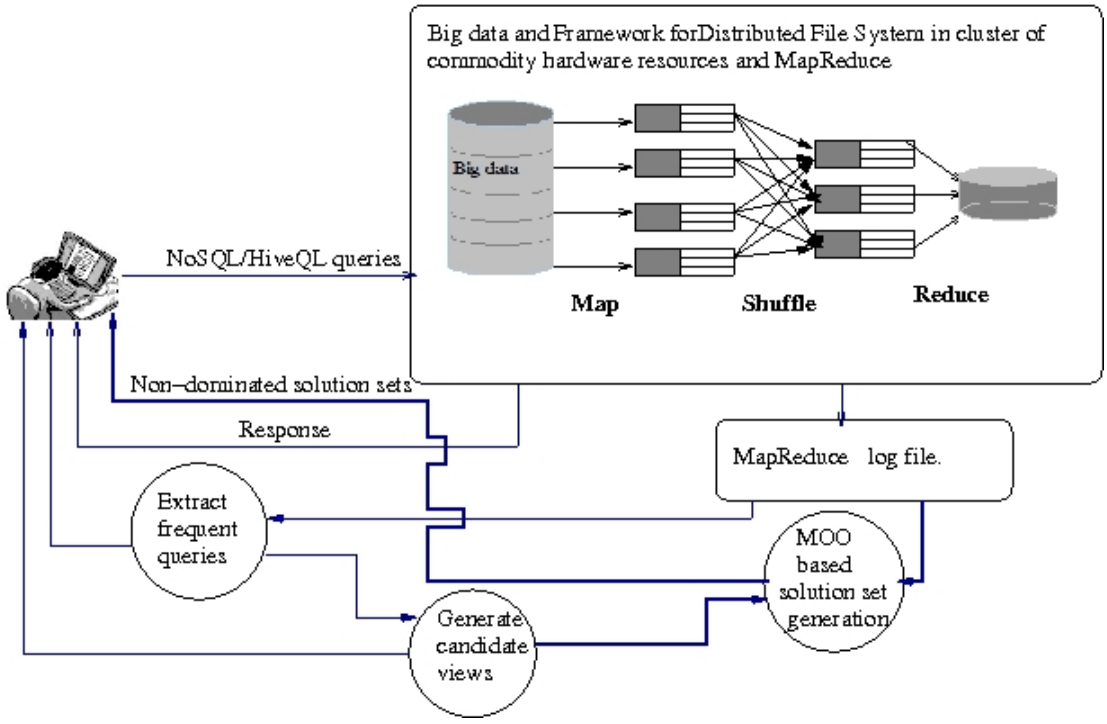

Figure 4-1: Test-bed for selecting non-dominated solution sets of materialized views

generated on processing HiveQL instructions, has been implemented. The recommended solution sets generated by both the implementations are analyzed. A set of HiveQL queries has been synthesized for triggering on data warehouse in a single node implementation of experimental HDFS. This set of queries considered as the set of frequent queries and are broken-up into some sub-queries or views which are considered as candidate views for our experimentation.

4.5.1 Experimental setup

In this experimental setup, Hortonworks Data Platform (HDP) version 2.0.6 has been used with Hortonworks Sandbox version 2.0 VMware for 64 bit CentOS operating system workstation 6.5-7.x virtual machine [83]. This is a single node implementation for experimenting HDFS. Hadoop version 2.2.0 and Hive version 0.12.0 of Apache [80] are used here, which let us manage data, perform ad-hoc queries and perform analysis of data warehouse in Hadoop cluster. For executing HiveQL queries, an HDP interactive interface to Hive^{TM} named Beeswax provided by HDP has been used. Using Beeswax we can type in HiveQL queries and have Hive evaluate them for us using a series of MapReduce jobs.

A block diagram of my test-bed is presented in Fig. 4-1. Though generally "Big data" means database of Tera byte/Peta byte size, HDP is designed for single node implementation for experimenting HDFS with smaller sized databases. In this experimentation Lahman Baseball Database of American Major League Baseball statistics from 1871 through 2011 [84] have been used as suggested by

HDP 2.0.6 for experimenting with Hadoop version 2.2.0.

For generating different cost function values for experimentation, few HiveQL queries on Lahman baseball database have been synthesized as listed below.

List of HiveQL test queries:

Q1. select a.yearid, a.playerid, b.average, c.totgs from batting_data a join (select yearid, playerid, avg(r) as average from batting_data group by yearid , playerid) b on a.playerid=b.playerid and a.yearid=b.yearid join (select yearid, playerid, sum(gs) as totgs from appearances_data group by yearid, playerid) c on a.playerid=c.playerid and a.yearid=c.yearid ;

Q2. select a.yearid, a.playerid, b.average, c.tot_batting from batting_data a join (select yearid, playerid, avg(r) as average from batting_data group by yearid, playerid) b on a.playerid=b.playerid and a.yearid=b.yearid left outer join (select yearid, playerid, sum(g_batting) as tot_batting from appearances_data group by yearid, playerid) c on a.playerid=c.playerid and a.yearid=c.yearid ;

Q3. select a.yearid, a.playerid, b.wild_pitches, c.tot_glf from fielding_data a join (select yearid, playerid, sum(wp) as wild_pitches from fielding_data group by yearid, playerid) b on a.yearid=b.yearid and a.playerid=b.playerid left outer join (select yearid, playerid, sum(glf) as tot_glf from fielding_of group by yearid, playerid) c on a.yearid=c.yearid and a.playerid=c.playerid ;

Q4. select a.yearid, a.teamid, a.playerid, b.average, c.wild_pitches from appearances_data a left outer join (select yearid, playerid, teamid, avg(r) as average from batting_data group by yearid, teamid, playerid) b on a.playerid=b.playerid and a.yearid=b.yearid and a.teamid=b.teamid left outer join (select yearid, playerid, sum(wp) as wild_pitches from fielding_data group by yearid, playerid) c on a.yearid=c.yearid and a.playerid=c.playerid ;

Q5. select a.lahmanid, a.playerid, a.hofid, b.yearid, b.totgs, c.totvotes from master_data a left outer join (select yearid, playerid, sum(gs) as totgs from appearances_data group by yearid, playerid) b on a.playerid=b.playerid left outer join (select hofid, yearid, sum(votes) as totvotes from hall_of_fame group by hofid, yearid) c on a.hofid=c.hofid and b.yearid=c.yearid;

Q6. select a.playerid, b.playedin, a.teamid, c.winner from appearances_data a join (select playerid, count(*) as playedin from appearances_data group by playerid) b on a.playerid=b.playerid left outer join (select teamidwinner, count(*) as winner from series_post group by teamidwinner) c on a.teamid=c.teamidwinner ;

Q7. select a.playerid, b.total_played, c.tot_fielding, d.tot_pitching from appearances_data a join (select playerid, sum(g_all) as total_played from appearances_data group by playerid) b on a.playerid=b.playerid left outer join (select playerid, sum(g) as tot_fielding from fielding_data group by playerid) c on

a.playerid=c.playerid left outer join (select playerid, sum(g) as tot_pitching from pitching_post group by playerid) d on a.playerid=d.playerid ;

Q8. select a.yearid, a.teamid, a.lgid, a.playerid, b.playedin, c.winner from appearances_data a join (select yearid, teamid, lgid, playerid, count(*) as playedin from appearances_data group by yearid, teamid, lgid, playerid) b on a.yearid=b.yearid and a.teamid=b.teamid and a.lgid=b.lgid and a.playerid=b.playerid left outer join (select teamidwinner , count(*) as winner from series_post group by teamidwinner) c on a.teamid=c.teamidwinner ;

Q9. select a.yearid, a.teamid, a.playerid, b.average, c.max_runs from appearances_data a left outer join (select yearid, playerid, teamid, avg(r) as average from batting_data group by yearid, teamid, playerid) b on a.playerid=b.playerid and a.yearid=b.yearid and a.teamid=b.teamid left outer join (select playerid , max(r) as max_runs from batting_data group by playerid) c on a.playerid=c.playerid;

Q10. select a.playerid, a.namefirst, a.namelast, b.max_runs, c.min_runs from master_data a join (select playerid , max(r) as max_runs from batting_data group by playerid) b on a.playerid=b.playerid join (select playerid, min(r) as min_runs from batting_data group by playerid) c on a.playerid=c.playerid ;

Q11. select a.playerid, a.hofid, b.runs_allowed, c.tot_votes from master_data a left outer join (select playerid, sum(r) as runs_allowed from pitching_post group by playerid) b on a.playerid=b.playerid left outer join (select hofid, sum(votes) as tot_votes from hall_of_fame group by hofid) c on a.hofid=c.hofid ;

Q12. select a.playerid, b.tot_leftfielding, c.namefirst, c.namelast from fielding_post a left outer join (select playerid, sum(g_lf) as tot_leftfielding from appearances_data group by playerid) b on a.playerid=b.playerid join (select playerid, namefirst, namelast from master_data) c on a.playerid=c.playerid ;

Q13. select a.playerid, b.tot_innouts, c.namefirst, c.namelast from fielding_post a join (select playerid, sum(innouts) as tot_innouts from fielding_post group by playerid) b on a.playerid=b.playerid join master_data c on a.playerid=c.playerid ;

Q14. select a.hofid, a.yearid, a.category, b.namefirst, b.namelast, c.batting from hall_of_fame a join master_data b on a.hofid=b.hofid join (select playerid, sum(g_batting) as batting from appearances_data group by playerid) c on b.playerid=c.playerid;

Q15. select a.playerid, a.bats, b.tot_runs, b.tot_hits from master_data a left outer join (select playerid, sum(r) as tot_runs, sum(h) as tot_hits from batting_post group by playerid) b on a.playerid=b.playerid ;

Q16. select a.playerid, b.tot_runs, c.tot_hits, c.bats, c.namefirst, c.namelast from batting_post a join (select playerid, sum(r) as tot_runs, sum(h) as tot_hits from batting_post group by playerid) b on a.playerid=b.playerid join master_data c on a.playerid=c.playerid ;

Q17. select a.playerid, b.tot_games, b.errors, c.throws, c.namefirst, c.namelast from fielding_post a join (select playerid, sum(g) as tot_games, sum(e) as errors from fielding_post group by playerid) b on a.playerid=b.playerid join master_data c on a.playerid=c.playerid ;

Q18. select a.playerid, a.glf, a.gcf, a.grf, b.tot_games, c.tot_outfielding from fielding_of a join (select playerid, sum(g) as tot_games from fielding_post group by playerid) b on a.playerid=b.playerid left outer join (select playerid, sum(g_of) as tot_outfielding from appearances_data group by playerid) c on a.playerid=c.playerid ;

Q19. select a.playerid, a.stint, a.yearid, a.teamid, a.lgid, b.r, b.h from batting_data a join (select * from batting_post where r>0) b on a.playerid=b.playerid and a.yearid=b.yearid and a.teamid=b.teamid and a.lgid=b.lgid ;

Q20. select a.playerid, a.yearid, a.teamid, a.lgid, b.g, b.r, b.h, b.rbi from appearances_data a join (select * from batting_post where r>0) b on a.playerid=b.playerid and a.yearid=b.yearid and a.teamid=b.teamid and a.lgid=b.lgid ;

The constituent views and aggregation functions that are to be considered as candidate views for materializing are extracted from these queries by a semantic analysis process. The queries and their constituent views are presented in Table 4.1. The queries and constituent views are triggered to HDP to get query processing costs, view processing costs and maintenance costs in terms of MapReduce CPU time along with the space requirements for the views. The HDP and Beeswax interface generates responses as well as log-files against HiveQL commands. These log-files contain all MapReduce split details and associated CPU costs along with the MapReduce jobs creation details. Different costs against these queries and views are extracted from log-files and stored in a database. The extracted costs of our queries in one instance of execution are presented in Table 4.2 and 4.3. The materialized view selection process takes input from this database for recommending optimum sets of views for materializing. All the selected frequent queries and candidate views are indexed and labeled to represent the solution vectors such that if the first dimension of the solution vector is 1, the first view is selected for materializing and if the second dimension of the solution vector is 0, the view labeled as second view is not to be selected for materializing and so on. This representation is used in many materialized view selection techniques [8,9,14,25]. The extracted costs of our queries in one instance of execution as we present in Table 4.2 and 4.3 are used as input to our multi-objective EA based view selection recommendation system.

4.5.2 Parameters used

In Differential Evolution (DE) algorithm based applications the main control parameters are the mutation scaling factor F, the solution population size NP and the cross-over parameter CR. In [28] it has been suggested that the value of NP

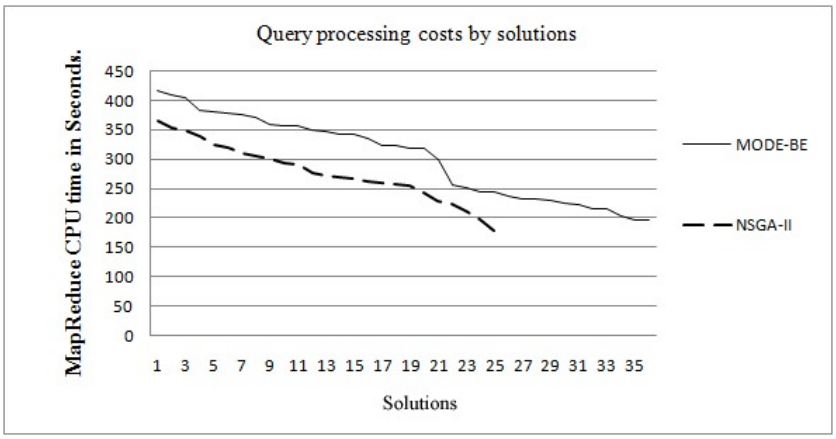

Figure 4-2: Processing MapReduce cost (in Seconds) by NSGA-II and MODE-BE generated non-dominated solutions.

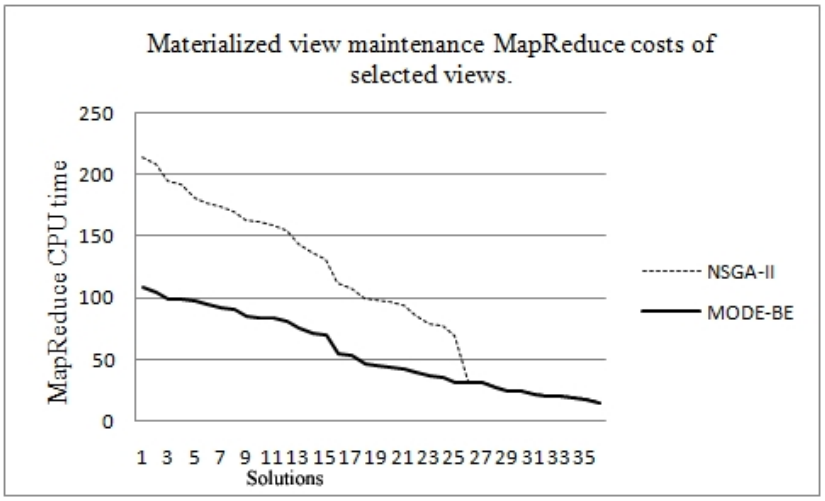

Figure 4-3: Materialized view maintenance MapReduce cost (in Seconds) by NSGA-II and MODE-BE generated non-dominated solutions.

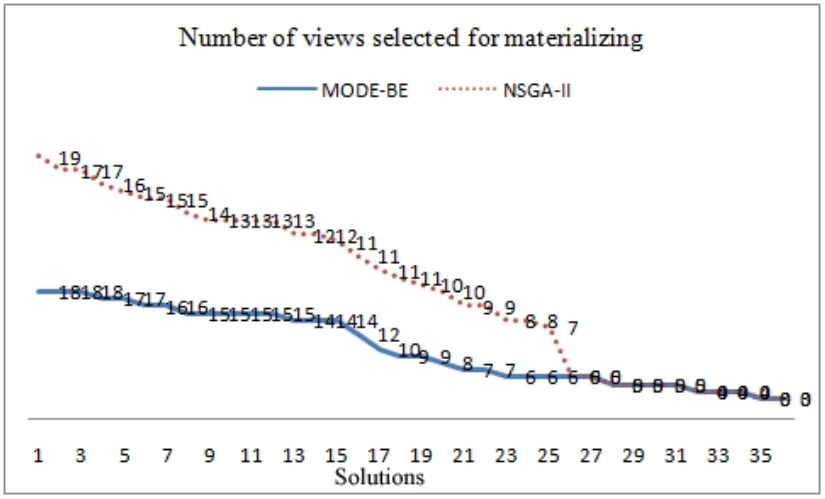

Figure 4-4: Number of views in solution sets for materializing.

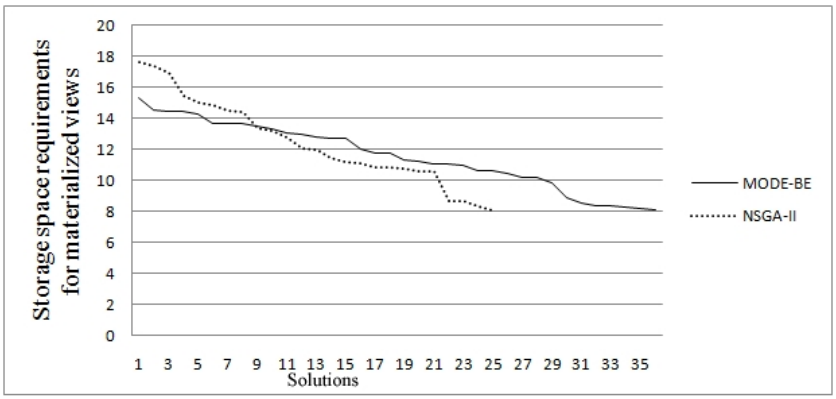

Figure 4-5: Space requirements by solution sets of views for materializing.

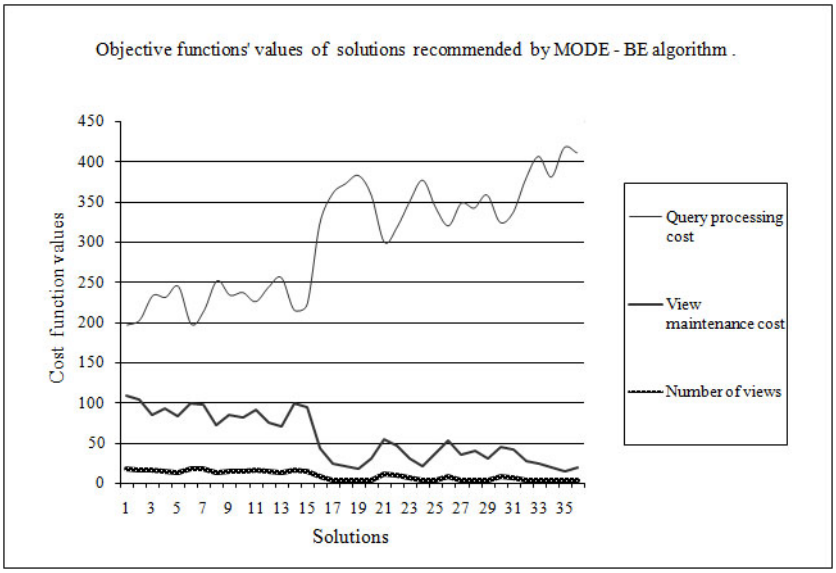

Figure 4-6: Objective functions' values by MODE-BE generated non-dominating solutions.

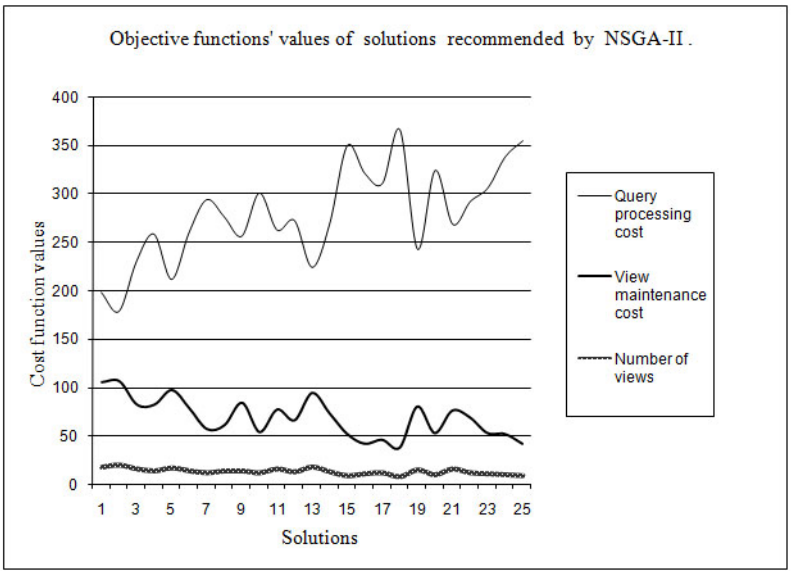

Figure 4-7: Objective functions' values by NSGA-II generated non-dominating solutions.

Table 4.1: Considered frequent HiveQL queries and constituent views

HiveQL queries	Constituent views
Q_1	v_1,v_2
Q_2	v_1,v_3
Q_3	v_4,v_5
Q_4	v_4,v_6
Q_5	v_2,v_7
Q_6	v_8,v_9
Q_7	v_{10},v_{11},v_{12}
Q_8	v_9,v_{13}
Q_9	v_6,v_{14}
Q_{10}	v_{14},v_{15}
Q_{11}	v_{16},v_{17}
Q_{12}	v_{18},v_{19}
Q_{13}	v_{20}
Q_{14}	v_{21}
Q_{15}	v_{22}
Q_{16}	v_{22}
Q_{17}	v_{23}
Q_{18}	v_{23},v_{24}
Q_{19}	v_{25}
Q_{20}	v_{25}

should be around 5 to 10 times the dimensionality of the problem. Therefore, for 25 candidate views, we may set values of NP between 125 to 250. In DE, a good choice of F is 0.5. The value of CR indicates number of inheritance by the mutant vector. According to [27, 73], for population size NP between 3 to 8 times of the dimensionality of the problem, the mutation scaling factor F=0.6 and cross-over ratio CR between 0.3 to 0.9 are good choices. In our multi-objective DE for binary represented decision variables (MODE-BE) we used the following parameters -

- the population size, NP=125,
- number of generations=50,
- selection scheme=*DE/rand/1/bin*,
- F=0.5,
- binary cross-over probability CR=0.3.

To compare the performance of MODE-BE and NSGA-II in materialized view selection problem for HDFS based data warehousing, we used following parameters with NSGA-II based system.

- the population size, NP=125,

Table 4.2: Query responding MapReduce costs of selected queries

HiveQL queries	HDFS MapReduce cost (in Seconds)
Q_1	45.05
Q_2	37.62
Q_3	37.35
Q_4	42.59
Q_5	25.51
Q_6	24
Q_7	25.48
Q_8	38.87
Q_9	29.77
Q_{10}	18.29
Q_{11}	22.57
Q_{12}	14.15
Q_{13}	12.68
Q_{14}	29.27
Q_{15}	13.83
Q_{16}	20.56
Q_{17}	22.04
Q_{18}	21.82
Q_{19}	2.39
Q_{20}	2.31

Table 4.3: Processing and maintenance MapReduce costs and space requirements of candidate views

Candidate view	Processing MapReduce cost (in Seconds)	Maintenance MapReduce cost (in Seconds)	Space (in MB)
v_1	11.52	4.023	2.2
v_2	16.34	4.234	2.236
v_3	19.43	4.034	2.151
v_4	14.16	2.652	2.2
v_5	8.37	6.051	0.267
v_6	13.85	13.042	2.9
v_7	6.84	11.521	0.0956
v_8	11.74	3.231	0.296
v_9	7.75	6.455	0.001
v_{10}	15.71	5.823	0.316
v_{11}	16.98	10.034	0.313
v_{12}	6.02	5.034	0.0252
v_{13}	20.12	4.611	3.252
v_{14}	11.05	6.810	0.3
v_{15}	13.76	5.430	0.294
v_{16}	6.61	4.517	0.026
v_{17}	10.73	2.220	0.021
v_{18}	10.11	4.315	0.297
v_{19}	1.84	5.412	0.519
v_{20}	7.28	3.021	0.061
v_{21}	16.38	5.011	0.314
v_{22}	8.07	9.025	0.075
v_{23}	6.99	7.25	0.07
v_{24}	11.58	9.312	0.299
v_{25}	2.3	5.812	0.487

- number of generations=50,
- Size of mating pool=125,
- tournament size=2,
- individual cross-over probability=1,
- individual mutation probability=1.

Two other problem specific parameters - maximum number of views that may be selected and minimum size of storage space to be used are also to be specified as well. These two parameters are mainly dependent on size of memory block size and split size of the HDFS. Generally HDFS block-size and split-size are of 64MB or 128MB. Therefore, as small files may use unnecessary MapReduce split and overhead, Hadoop works better with smaller number of larger files [17].

4.5.3 Results and observations

In my experimentation with the above mentioned experimental setup, parameters and data, it has been observed that,

- the NSGA-II based system converges more quickly than MODE-BE based recommendation system. But, as for preserving diversity among solutions in MODE-BE, distances among solutions in their solution vector space are used instead of crowding-distance in objective function space, the standard deviation between solutions generated by MODE-BE is 5.203402 whereas that of NSGA-II is 3.120897. The diversity in solution vector space is preferred because diversity preservation on objective function values may lead to loss of some significantly distinct solutions on the basis of constituent selected views in them. This may obviously happen because a scalar valued function with different vector parameters may result same scalar value.
- In our experimentation it can be observed that MODE-BE generates 37.04% more number of solutions than NSGA-II based system. More number of solutions with comparable quality of solutions may be useful for selecting most appropriate solutions depending on user application requirements. For filtering significant solutions from the obtained solutions, distances in solution vector space for each solution to all other solutions yielded may be computed and then based on the mean (μ) distances and their standard deviation (σ), filtration criteria may be applied [25].
- From the query processing costs in terms of MapReduce time, as plotted in Figure 4-2, it can be observed that solutions generated by MODE-BE are more costly in case of query processing and responding than of NSGA-II generated solutions. But, the MapReduce time for maintaining the materialized views are less in case of MODE-BE.

- From Figure 4-4 and 4-5 it can be observed that MODE-BE results are slightly better based on the Hadoop framework's basic criterion that lesser number of bigger views or tables are to be considered for materializing. For our experimental data presented here, we found that the minimum size of storage space requirement is slightly more in case of MODE-BE.
- Again, by applying Mann-Whitney U test on both MODE-BE and NSGA-II generated solutions at 5% level of significance, i.e at $\alpha = 0.05$, we cannot reject the null hypothesis that the solution vectors generated by both the systems are from the same population.

4.6 Discussion

I have presented here my study on view selection techniques for materializing in distributed commodity hardware file system data warehousing. The main contribution here is establishing the view selection for materializing problem in distributed commodity hardware file system as a multi-objective optimization problem and study of performances by multi-objective Differential Evolution algorithm and NSGA-II for solving this problem. As it is an NP-hard problem, I used multi-objective evolutionary algorithm based approach for designing a recommendation system for selecting views for materializing.

A prototype of view selection framework has been designed that uses log-files generated by single node implementation of HDFS based data warehousing framework and Hive 0.12.0 queries. As the approach is a generic one, it may be implemented easily for any kind of similar framework that uses distributed cluster of commodity hardware.

Though, in this experimental framework, the cost functions, number of views in different solution sets and space costs of each individual solutions are computed by the recommending system, the semantic analysis process for extracting and composing candidate views is yet to be developed. At present the candidate views are fed to the HDP by an offline process for computing different associated costs.

The present popular version of Hive^{TM} does not support materialized views. The work presented here leads towards efficient Big data query processing by materializing selected views recommended by the system in cluster of distributed commodity hardware file system. There may be alternative technologies for implementing the materialized view selection and/or selection of response of queries for in-memory materialization like Discardable In-Memory Materialized Query (DIMMQ), Spark's Resilient Distributed Data-set (RDD)[19] etc.. We believe that community like Hadoop may find many more such technologies and will suggest bench-mark framework for unbiased evaluation of different techniques for efficient Big data query processing.

In this chapter the solution quality of multi-objective DE and NSGA-II in

view selection for materializing in Big data DFS framework have been analyzed based on the cost function values by the yielded solutions. The performances of non-deterministic multi-objective optimization techniques may be studied in different ways. In next chapter another widely used stochastic method for hard optimization problems called Simulated Annealing (SA) has been customized for solving this problem and comparative performances of MODE-BE, NSGA-II and multi-objective SA have been reported while applying in this problem based on measures on how the obtained solutions converge towards estimated true Pareto front, number of obtained solutions that are not dominated by solutions generated by other algorithms and distribution of solutions on the found Pareto front.