Multi-Objective Big Data View Materialization Using Improved Strength Pareto Evolutionary Algorithm

Akshay Kumar, Jawaharlal Nehru University, India
T. V. Vijay Kumar, Jawaharlal Nehru University, India*

ABSTRACT

Big data refers to the enormous heterogeneous data being produced at a brisk pace by a large number of diverse data generating sources. Since traditional data processing technologies are unable to process big data efficiently, big data is processed using newer distributed storage and processing frameworks. Big data view materialization is a technique to process big data queries efficiently on these distributed frameworks. It generates valuable information, which can be used to take timely decisions, especially in cases of disasters. As there are a very large number of big data views, it is not possible to materialize all of them. Therefore, a subset of big data views needs to be selected for materialization, which optimizes the query response time for a given set of workload queries with minimum overheads. This big data view materialization problem, having objectives minimization of the query evaluation cost of a set of workload queries, while simultaneously minimizing the update processing costs of the materialized views, has been addressed using improved strength pareto evolutionary algorithm (SPEA-2) in this paper. The proposed big data view selection algorithm, which is able to compute a set of diverse non-dominated big data views, is shown to perform better that existing big data view selection algorithms..

KEYWORDS

Big Data, Evolutionary Algorithm, Multi-Objective Optimization, SPEA-2, View Materialization

1 INTRODUCTION

Advances in Information and Communication Technology (ICT) have impacted almost every aspect of human life including health, education, commerce, agriculture, scientific exploration etc., triggering the generation of large amounts of data. This Big data has a very large volume; is produced by variety of sources; is generated at a high speed or velocity; generally has low veracity or trustworthiness; and has low value. These characteristics of Big data are referred to as the five V's of Big data (Jacobs, 2009; Zikopoulos et al., 2011; Gupta et al., 2012; Gandomi & Haider, 2015; Kumar & Vijay Kumar, 2015). Big data cannot be processed efficiently by the traditional technologies. This led to the emergence of Big data processing frameworks, which entail distributed processing over a Distributed File System (*DFS*). Some of the technologies and frameworks that can be used to process Big data include Hadoop, Apache Hadoop, map-reduce framework, *NoSQL* database, Apache Spark etc. (Hadoop 2008; Hadoop 2012; Manyika 2011; Dezyre, 2015; Dean & Ghemawat, 2012; Kumar & Vijay Kumar 2021a). These frameworks provide features of redundant Big data storage along with reliable distributed processing of big data.

DOI: 10.4018/JITR.299947 *Corresponding Author

Big data processing has the potential to provide useful, unforeseeable information, which can benefit society in many different ways. For example, healthcare systems generate large amount of clinical, diagnostic, medical imaging, and public health data received from large number of hospitals and health centers, which can be used to predict and monitor the spread or outbreak of infectious diseases (Luo et al., 2016). One of the most recent applications of Big data involves the determining of the extent and possible future spread of corona virus disease, called COVID19, which is threatening human health internationally. This Big data application has faced the technological challenges of integrating redundant data from multiple diverse data sources and processing such geographically spread Big data in real time (Zhou et al., 2020). Another interesting application of vig data has been proposed in (Bibri, 2018), which relates to the use of the Internet of Things (IoT) devices in smart cities. IoT devices can produce large amounts of Big data in smart cities, which relate to people's health, water system, electrical appliances, vehicles, machines, plants, soil, air etc. This Big data can be processed to determine the environmental impact of the smart cities, which can be used to model environmentally sustainable cities (Bibri, 2018). Thus, Big data applications can be created to facilitate healthcare, disease control, resource management, environmental protection etc. A Big data application is required to collect, clean, integrate, store, process, analyze and present information in various visual forms. Further, it must generate accurate real time information, as incorrect or delayed information has no value, especially in case of disasters. Thus, a Big data application must process data efficiently to produce information, which can be used for making timely decisions.

Big data view materialization enhances the efficiency of processing Big data queries over a Big data application. As large numbers of queries can be made on a Big data application, there could be a large number of possible Big data views. All such Big data views cannot be materialized, as that would result in a significant increase in the total size of Big data in an application. In addition, it will also increase the cost of update processing, as updates will be required to be performed on the Big data and the related materialized views. Therefore, a subset of Big data views must be selected for materialization, which optimizes the performance of Big data query processing. The view selection problem in the context of a data warehouse, has been shown to be an *NP-hard* problem (Harinarayan et al., 1996). This problem becomes more complex in the context of Big data.

The problem of selection of Big data views for materialization is concerned with selecting views that optimize the cost of processing of a set of workload queries. This optimization problem should incorporate Big data characteristics. (Kumar & Vijay Kumar, 2021b) presented the Big data view materialization, as a bi-objective optimization problem, with the objectives being the minimization of query evaluation cost and the minimization of the update processing cost with a constraint on the total available size for materialization. One of the ways to address this problem is by using the multi-objective evolutionary algorithms. (Kumar & Vijay Kumar, 2021b) used the Vector Evaluated Evolutionary Algorithm (*VEGA*) to solve this bi-objective optimization problem. In this paper, this bi-objective Big data view materialization problem is addressed using the Improved Strength Pareto Evolutionary Algorithm (*SPEA-2*) (Zitzler et al., 2001). *SPEA-2* follows an elitist approach and produce a set of non-dominated solutions to the multi-objective optimization problems. *SPEA-2* has features of fast convergence due to its elitist archive selection; less complexity due to use of fewer number of configuration parameters; creation of diverse solutions due to the use of density computations; and the production of non-dominated solutions. In this paper, a *SPEA-2* based Big data view selection algorithm to select non-dominated Big data views for materialization is proposed.

The paper is organized as follows: Section 2 presents the view materialization problem with focus on Big data view materialization. The bi-objective Big data view materialization problem, given in (Kumar & Vijay Kumar, 2021b), is briefly discussed in Section 3. Section 4 presents a *SPEA-2* based Big data view selection algorithm to solve the bi-objective Big data view materialization problem. An example illustrating the use of the proposed algorithm is given in Section 5 followed by experimental results in section 6. Section 7 is the conclusion.

2 VIEW MATERIALIZATION

The problem of selection of views for materialization was first studied in the context of a database management system. A view, in this context, is defined as a derived relation, which is produced as a result of a database query. Further, a materialized view stores data along with its definition. The problem of selection of views for materialization involves the selection of subsets of views for materialization that optimize the query response time for a given set of workload queries. As data in the database relations receive updates, the materialized views are also required to be maintained. (Gupta & Mumick, 1995) defined the issues of the view maintenance problem, viz. representation of the view creation query, identification of access rights on the view data, and the process of performing incremental updates on the views. (Ross et al., 1996) suggested that the maintenance cost of the materialized views can be optimized by identifying and maintaining additional views, which can be identified from the common query sub-expressions among the views. Further, in database systems, the materialized views can be updated incrementally to reduce maintenance costs. However, in case of Big data, such optimization is a tedious process, as Big data is voluminous and mostly unstructured.

View materialization in data warehouses is a widely studied problem and has been presented and solved using different approaches. (Roussopoulos, 1998) highlighted the potential of materialized views in query evaluation. The data in the data warehouse is represented as multi-dimensional data cubes. A cell in these data cubes represents a level of data aggregation, or a possible view, and the interrelationships among these views are represented by a lattice framework (Harinarayan et al., 1996). (Gupta, 1996) used an AND-OR view graph structure to represent the interrelationships amongst the views in a data warehouse. Selection of views for materialization is required to compute the query processing cost of workload queries and the maintenance cost of the materialized views. (Harinarayan et al., 1996) used the number of records that are processed to answer a query for computing the query processing cost. (Chirkova et al., 2001) presented a theoretical framework for view materialization for a data warehouse, which used estimates of view statistics generated by the query optimizers, for computing the query processing cost. (Mistry et al., 2001) suggested the optimization of maintenance cost by materializing common sub-expressions among the query expression graphs of the queries. Many different types of approaches have been designed for solving the view materialization problem for data warehouse. (Harinarayan et al., 1996; Gupta, 1996) used greedy approaches to select a set of views for materialization. (Agrawal et al. 2000) designed an empirical approach for automatic selection of materialized views and indexes from SQL queries and related statistics of query evaluation. (Mami et al., 2012) presented a survey of various view selection algorithms and also illustrated a classification of these algorithms. View selection problem in the context of the data warehouse has been solved using metaheuristic algorithms (Arun & Vijay Kumar, 2015a, 2015b, 2017a, 2017b; Vijay Kumar & Arun, 2016, 2017, Vijay Kumar & Kumar, 2014, 2015; Kumar & Vijay Kumar, 2018). The same problem was formulated as a bi-objective view selection problem and solved using multi-objective evolutionary algorithms *VEGA* (Prakash & Vijay Kumar, 2019a), *MOGA* (Prakash & Vijay Kumar, 2020a), *SPEA-2* (Prakash & Vijay Kumar, 2019b) and *NSGA-II* (Prakash & Vijay Kumar, 2020b).

View Materialization for Object-Oriented Database Management System (*OODBMS*) requires multiple and dynamic re-classification of objects into materialized view classes. A materialized view in *OODBMS* can be created without making a physical copy of the objects into the materialized view classes (Kuno et al., 1995), thus, minimizing the maintenance overheads of the views. View materialization has also been studied for influencing data intensive web applications in (Labrinthis & Roussopoulos 2000; Kumar & Vijay Kumar, 2020). A web page consists of many smaller components, called web-views where the web-view can be shared by several web pages. Materializing the web-views dynamically, with every access of a web page and every update of a stored relation, optimizes the processing of web page requests (Labrinthis & Roussopoulos 2000).

View materialization for semi-structured databases use query graphs to identify sub-trees from the ordered tree of semi-structured data (Abiteboul et al., 1997; Tang et al., 2009). (Abiteboul, 1999)

presented a three-level view architecture for XML data, namely the database server level, view server level and the display component level. (Tang et al., 2009) presented a greedy algorithm to select views for materialization on semi-structured XML data. The view maintenance problem on semi-structured data is handled through an incremental update model (El-Sayed et al., 2006). View materialization has not been specifically studied for unstructured data, eventhough, efficient handling of such data is essential to support view materialization on Big data, since it is the largest segment of Big data. The basic characteristics of unstructured data have been presented in (Gandomi & Haider, 2015), which also illustrates the methods of processing unstructured data by including data integration from heterogeneous resources. (Yafooz et al., 2013) proposes three techniques for processing queries on unstructured data, viz. linking the unstructured data with the relational model and/or creating models on unstructured data and/or extracting data using natural language queries. (Lu et al., 2017) proposes use of a hybrid architecture to enhance the efficiency of batch processing jobs on unstructured data using the map-reduce framework.

In general, view materialization for different types of database systems involves the optimization of the query response time and the view maintenance costs with a constraint on the cumulative size of the materialized views. Further, the problem of selection of views for materialization is solved primarily for the structured data. However, Big data view materialization is concerned with the materialization of views from structured, semi-structured and unstructured data considering the characteristics of Big data. Big data view materialization is discussed next.

2.1 Big Data View Materialization

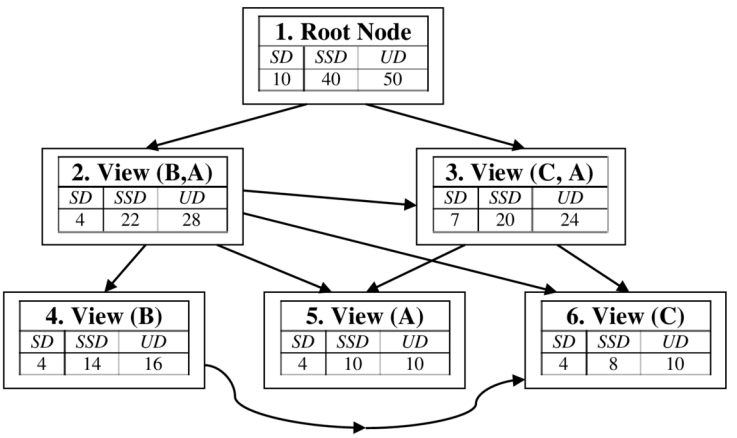
Big data view materialization aims to optimize the processing time of Big data queries. It should also address issues relating to the storage and processing of large heterogeneous data, maintenance of materialized views, rapid rate of data generation having low data integrity. (Goswami et al., 2017) studied Big data view materialization for Hive, a Big data warehousing tool, for a specific dataset. It defined view materialization, as a multi-objective problem and solved it using the differential evolution algorithm and non-dominated sorting genetic algorithm. However, their approach to Big data view materialization is an extension of the view materialization approach on data warehouses and does not consider the Big data characteristics for the computation of query processing cost and view maintenance cost. (Kumar & Vijay Kumar, 2021a) presented the Big data view materialization problem, which considers Big data characteristics like data heterogeneity, data volume, data integrity, data generation rate; and query frequencies to compute the total query processing cost of the workload queries. (Kumar & Vijay Kumar, 2021a) also presented a greedy algorithm to solve the problem of selecting Big data views for materialization. (Kumar & Vijay Kumar, 2021b) presented the Big data view materialization problem defined in (Kumar & Vijay Kumar, 2021a), as a bi-objective optimization problem, with the two objectives being the minimization of the query evaluation cost for a given set of workload queries within a specific time window; and the minimization of the update processing cost for such materialized Big data views; with a constraint on the total size of the materialized views. (Kumar & Vijay Kumar, 2021b) used the vector evaluated genetic algorithm (*VEGA*) to solve the Big data view materialization problem.

The views on Big data are generated as a result of a query. For identifying the candidate Big data views for materialization, query attributes (Q_A) are used. Query attributes (Q_A) are those attributes of Big data queries that results in the retrieval of structured data or the extraction of semi-structured and unstructured data (Kumar & Vijay Kumar, 2021a). These query attributes could have query attribute dependencies. For example, a query attribute dependency $B \rightarrow C$ indicates that the Big data created based on a query involving query attribute *B* can derive the Big data created based on the query involving query attribute *C.* In other words, a Big data view created using query attributes B, can answer the query that involves query attributes C. Thus, the query attributes and their dependencies form a complex interrelationship structure, which can be represented using a directed graph (Kumar & Vijay Kumar, 2021b). Figure 1 shows an example of the view relationship graph, which has been

created using three query attributes (Q_A) {*A*, *B*, *C*} with a query attribute dependency *B*→*C*. Each node, called a view node, in Fig. 1 represents a view consisting of structured, semi-structured and unstructured data. Also, a view node is created for a specific subset of query attributes. The links in Fig. 1 represent a view dependency, i.e. a link from view node *X* to view node *Y* indicates that a query that can be answered by the view node *Y* can also be answered by view node *X*. A query attribute dependency also results in the creation of links between the view nodes. For example, in Fig. 1, a query attribute dependency *B*→*C* generates links between the view nodes *View(B)* to *View(C)*, *View(B, A)* to *View(C, A)* and *View(B, A)* to *View(C)*. In addition, query dependency may result in the removal of certain nodes from the graph. For example, the view node *View(B, C)* is same as *View (B)* due to the query attribute dependency *B*→*C*, therefore, view node *View(B, C)* is not included in the graph in Fig. 1.

It may be noted that several views can be generated from every view node of Fig. 1. A view, so generated, may not involve all types of data of the view node, e.g. a view may be created on the structured and semi-structured data from view node *View(B, C)*. In addition, the three views of the root node (Fig. 1), one each for structured, semi-structured and unstructured data respectively, are always assumed to be materialized. Figure 1 can also help in computing alternate query evaluation plans. For example, a query, which can be answered by a view created from view node *View(C)* can also be answered by views created from view nodes *View(B)*, *View(C, A)* or *View(B, A)*, provided all these views are created from similar data.

Figure 1. Structure for Big Data views based on QA{A, B, C} with dependency B→C



SD-Structured Data, *SSD*-Semi-structured Data, *UD*-Unstructured Data

The frequencies of the queries in a Big data application are dynamic in nature and hence, Big data view materialization is defined for a specific window of time. Thus, selection of views for materialization on Big data may be performed periodically, during the lifetime of a Big data application.

A bi-objective Big data view materialization problem given in (Kumar & Vijay Kumar, 2021b) is briefly discussed next.

3 BI-OBJECTIVE BIG DATA VIEW MATERIALIZATION PROBLEM

Big data view materialization aims at minimizing the query evaluation cost while simultaneously minimizing the cost of processing updates on the view data. Further, it has been defined as a bi-objective optimization problem with the objectives being the minimization of the query evaluation cost for a set of workload queries for a specific time window; and the minimization of the update processing cost of materialized Big data views subject to a constraint on the total size of the Big data views (Kumar & Vijay Kumar, 2021b). These objectives are briefly discussed below:

Minimize Query Evaluation Cost (QEC_{BDV}) for a given set of workload queries and their frequencies during a specific time window:

The query evaluation cost (QEC_{BDV}) is the cost of evaluating the workload queries using a set of materialized views. Each of these queries can be computed using alternate query evaluation plans. Big data views need to be selected for materialization in order to minimize the total query evaluation cost of the workload queries. QEC_{BDV} is computed using the query frequencies along with the minimum cost of evaluating a query (MCQ_i). The minimum cost of evaluating the i^{th} query (MCQ_i) for a given set of materialized views using the alternative query evaluation plans for that query is given by (Kumar & Vijay Kumar, 2021b):

$$MCQ_i = Min_j \left[\sum_k \left\{ m_k \times CMV_k \times \left(1 + \frac{1}{2} \times ufv_k \right) \right\} + \left\{ \left(1 - m_k \right) \times CV_k \times \left(1 + \frac{1}{2} \times uf_k \right) \right\} \right] \quad (1)$$

In the above equation, MCQ_i is computed by finding the minimum cost of a query evaluation from amongst the *j query* evaluation plans. A query evaluation plan can consist of *k* views and for each query evaluation plan, the cost of evaluating a query is computed using the number of stored Big data blocks of k^{th} view (CMV_k), in case the k^{th} view is materialized; or the number of stored Big data blocks of data that is required to compute the k^{th} view (CV_k), in case the k^{th} view is not materialized. The variable m_k is a binary variable and the value of m_k is 1, if the k^{th} view is materialized, or 0, if the k^{th} view is not materialized. The minimum of these query evaluation costs from all the *j* query evaluation plans, is the minimum cost of evaluating the query (MCQ_i) for the given set of materialized views. The ufv_k and uf_k are the size increase factor for the view and the data from which the view is computed respectively. The computation of the query evaluation cost of all the workload queries (QEC_{BDV}) is given by (Kumar & Vijay Kumar, 2021b):

$$QEC_{BDV} = \sum_{i=1}^{n} \left(MCQ_i \times f_i \right) \quad (2)$$

In the above equation, f_i is the query frequency of the i^{th} query and *n* represents the number of queries.

Minimize the update processing cost (UPC_{BDV}) of the Big data views selected for materialization:

The update processing cost (UPC_{BDV}) of the materialized views is computed using the size of data, in terms of Big data blocks, which is required to be processed to update a Big data view, along with the integrity factor of the view (Kumar & Vijay Kumar, 2021b). The integrity of different types of data is different. For example, integrity of unstructured data is lower than that of the semi-structured or structured data. Low integrity results in generation of additional data for updating a view. The computation of the update processing cost of all the Big data views that are materialized is given by (Kumar & Vijay Kumar, 2021b):

$$UPC_{BDV} = \sum_i \left(m_i \times UMV_i \div I_{m_i} \right) \quad (3)$$

In the equation (3), UMV_i is the cost of data that is to be processed to update the i^{th} materialized view and I_{mi} is the integrity factor related to the i^{th} Big data view. The variable m_i is a binary variable, which takes a value 1, if the i^{th} Big data view is materialized, else it takes a value 0.

It may be noted that CMV_k, CV_k and UMV_i, which are used for computing QEC_{BDV} and UPC_{BDV}, are represented in terms of the size of the stored Big data blocks (Kumar & Vijay Kumar, 2021a). The computation of *QEC* and *UPC* using Big data blocks is more reliable and consistent in comparison to performing the computation using the number of records, which are significant in case of Big data; or performing the computation using the time of computation of queries and views, which is dependent on the number and state of the map-reduce nodes used for computation.

The above two objectives are conflicting in nature; accordingly, evolutionary algorithms can be applied to solve the above bi-objective Big data view selection problem. In this paper, an Improved Strength Pareto Evolutionary Algorithm (Zitzler et al., 2001), i.e. *SPEA-2*, has been used to solve the bi-objective Big Data View Selection Problem (Kumar & Vijay Kumar, 2021b). Big data view selection using *SPEA-2* is discussed next.

4 BIG DATA VIEW SELECTION USING *SPEA-2*

Strength Pareto evolutionary algorithm (*SPEA*) was proposed in (Zitzler et al., 1999). *SPEA* is an elitist multi-objective evolutionary algorithm, which stores elitist solutions explicitly as an archived population. (Zitzler et al., 1999; Zitzler et al., 2000) illustrated that *SPEA* performs well in comparison to other multi-objective genetic algorithms, although, it had several limitations. These limitations were related to assigning a fitness value to a solution, the estimation of the density of the solutions and the truncation of the solutions when non-dominated solutions exceed the archive size (Zitzler et al., 2001). *SPEA-2* proposed in (Zitzler et al., 2001) addressed these limitations and has performed well in comparison to other multi-objective genetic algorithms. Further, it produces the Pareto optimal solutions. Therefore, Big data view selection for materialization, which is a bi-objective optimization problem, has been solved using *SPEA-2* (Zitzler et al., 2001), which is discussed next.

4.1 *SPEA-2*

Strength Pareto evolutionary algorithm (*SPEA-2*) was proposed in (Zitzler et al., 2001). In *SPEA-2,* initially, a random population of size *N* is generated and the archived population is initialized to null. The strength of a solution in *SPEA-2* is defined as the number of solutions in the population and the archived population that are dominated by it. *SPEA-2* computes the raw fitness of each solution, as the sum of the strength of the solutions in the population and the archived population that dominates that solution. Thus, a raw fitness value 0 for a solution implies that no other solution dominates it. The density of solutions is determined by measuring the *k*-nearest distance from the other solutions. A higher density implies that a solution has closer neighbors. The raw fitness and density are used to compute the fitness value of a solution. The non-dominated solutions are selected from the population and the archived population to create the next generation archived population. *SPEA-2* ensures that the size of the archived population is fixed. Thus, if the number of non-dominated solutions is less than the size of the archived population, the best dominated solutions are added to the archived population of the next generation; else if the non-dominated solutions are more than the size of archived population, the extra solutions are truncated from the archived population of the next generation, using the density value. The mating pool is created using the binary tournament selection with replacement from the archived population of the next generation. The crossover and mutation are performed to create the population for the next generation. The algorithm is run for a specified number of generations and the non-dominated solutions of the last generation forms the output of the genetic algorithm.

In the next sub-section, a Big data view selection algorithm using *SPEA-2* ($BDVSA_{SPEA-2}$) is presented.

4.2 Big Data View Selection Algorithm ($BDVSA_{SPEA-2}$)

$BDVSA_{SPEA-2}$ is an evolutionary algorithm, based on *SPEA-2*, to solve the bi-objective view materialization problem discussed above. The algorithm is presented below:

The input to $BDVSA_{SPEA-2}$ algorithm are - the list of frequent queries (Q) over a time window; the frequencies of these queries during the same time window (f); the list of Big data views (CVV_{DB}), which are candidates for view materialization identified using query attributes (Q_A) and queries (Q); mapping of queries to alternate query evaluation plans (*QtVs*), which are created using Big data views (CVV_{DB}) and the view structure graph (Fig. 1); the cost matrix of views, which includes the cost of the data, in terms of Big data blocks, required to compute the view (*CV*) and the cost of materialized views (*CMV*), in terms of Big data blocks; the update factors - *ufv*, which represents the change in the size of the view compared to the original size of the view, and *uf*, which represents the change in size of the data used to create the view to its original size; size of the data, in terms of Big data blocks, required to be processed to update a materialized view (*UMV*); and the Integrity Factor (I_m), which represents the correctness of data in each candidate Big data view. The chromosome for the algorithm is a linear view vector of materialized Big data views of size k (*VVk*). Figure 2 shows the structure of the chromosome of size 4, i.e. a view vector of size 4 (*VV4*). Each element in this view vector is a Big data view identified by a view identifier (*Vid*). The value of k, which represents the number of views to be materialized, also acts as an input to the algorithm.

Figure 2. The View Vector of size 4 (VV4)

Vid_1	Vid_2	Vid_3	Vid_4

$BDVSA_{SPEA-2}$
Bi-objective Problem: Minimize the query evaluation cost (QEC_{BDV}); and minimize the update processing cost (UPC_{BDV}) for a given query workload over a time window.
Input
 List of frequent Queries (Q) and their frequencies (f) in a time window
   List of Candidate Big data views for materialization (CVV_{BD})
   Mapping of queries to alternate query evaluation plans using Big data views ($QtVs$)
   Cost vectors of Candidate Views (number of Big data blocks)
      Size of materialized view (CMV), and
      Size of Data, which is used for computing the view (CV)
   Update factor of big data required to compute i^{th} View V_{BDi} (uf_i)
   View Integrity Factor (I_m)
   Data size, in terms of Big data blocks, for updating i^{th} materialized view(UMV_i)
      The size of view vectors, representing number of views to materialize (k)
Output
   Set of non-dominated view vectors of size k
Procedure:
Initialize:
   Population Size (N); Archive size ($Narc$); Number of Generations (n_g);
   Probability of crossover (p_c); Probability of mutation (p_m);

Step 1: Generate Initial Population:

Create a random population (*POna*) of size *N* of view vectors of size *k (VVk)*;

Create an empty archived population (*POarc*) = ∅ ;

Perform the following Steps WHILE generation $t <= n_g$

Step 2: Compute Objective function values

For every view vector (*VVk*) of population $VVk_{na} \in POna$

for every Query *i* in Q

Compute the *Minimum cost* of i^{th} Query MCQ_i using equation given below, by computing cost of query evaluation for every j^{th} query evaluation plan, where a plan consists of *z* views.

$$MCQ_i = Min_j\left[\sum_z\left\{m_z \times CMV_z \times\left(1+\frac{1}{2}\times ufv_z\right)\right\}+\left\{\left(1-m_z\right)\times CV_z \times\left(1+\frac{1}{2}\times uf_z\right)\right\}\right]$$

// MCQ_i is computed for each query

Compute the objective function values using the equations given below:

$$QEC_{BDV} = \sum_{i=1}^{n}\left(MCQ_i \times f_i\right)$$

$$UPC_{BDV} = \sum_i\left(m_i \times UMV_i \div I_{m_i}\right)$$

// compute for every VVk_{na}

Step 3: Assign Fitness

Compute the strength value (*StVVk*) for each view vector (VVk_i) ∈ {*POna* ∪ *POarc*} using the equation given below:

$$StVVk_i = \left|\left\{VVk_j \mid VVk_j \in \left\{POna \cup POarc\right\} \wedge VVk_i \succ VVk_j\right\}\right|.$$

The symbol:|…| specify cardinality of the set; ∪ finds the union of multi-set; and ≻ is a symbol for Pareto dominance relation (Zitzler et al., 2001).

Compute the raw fitness (Rf_i) for each VVk_i using the equation given below:

$$Rf_i = \sum_{\left[\left(j \in \left\{POna \cup POarc\right\}\right) \wedge \left(VVk_j \succ VVk_i\right)\right]} \left(StVVk_j\right)$$

Compute the fitness (Ff_i) for each VVk_i ∈ {*POna* ∪ *POarc*} as:

Compute the density (den_i) of each VVk_i using *k*-nearest method using equation given below:

$$den_i = \frac{1}{\left(\pm^{k}_{VVk_i} + 2\right)}$$

Here, $\pm^{k}_{VVk_i}$ is computed using Euclidian distance of the i^{th} view vector from all other view vectors VVk_j ∈ {*POna* ∪ *POarc*} using the equation given below:

$$dist_{ij} = \sqrt{\left(\mathrm{n}QEC_i - \mathrm{n}QEC_j\right)^2 + \left(\mathrm{n}UPC_i - \mathrm{n}UPC_j\right)^2}$$

These distances are then sorted. The k^{th} distance is used for computation of den_i. The value of *k* for the *k*-nearest neighbor method is $\sqrt{\left(N + Narc\right)}$

Compute the fitness (Ff_i) using the equation given below:

$$Ff_i = Rf_i + den_i$$

Step 4: Selection of Environment

Transfer all the non-dominated view vectors from *POna* and *POarc* to archived population of next generation using the equation given below:

$$\left(POarc\right)_{t+1} = \{VVk_i \mid VVk_i \in \left\{\left(POna\right)_t \cup \left(POarc\right)_t\right\} \wedge Ff_i < 1\}$$

Compare the |($POarc$)$_{t+1}$| with Archive size ($Narc$)
if |($POarc$)$_{t+1}$| == ($Narc$) the step is complete;
else if |($POarc$)$_{t+1}$| < ($Narc$)
Copy [($Narc$) - |($POarc$)$_{t+1}$|] best dominated view vectors
from *POna* and *POarc* to archived population of next generation
else if |($POarc$)$_{t+1}$| > ($Narc$)
Truncate |($POarc$)$_{t+1}$| - ($Narc$) view vectors one by one from archived population of next generation, which are closest to other view vectors.

Step 5: Termination:

if the generation $t >= n_g$ then collect all the non-dominated view-vectors of ($POarc$)$_{t+1}$.

Output the final non-dominating view-vectors from ($POarc$)$_{t+1}$.

Step 6: Mating Selection

Use binary tournament selection method with replacement on($POarc$)$_{t+1}$ to create mating pool:

Perform the following selection till mating pool is full
Select a binary pair of *VVk* from the ($POarc$)$_{t+1}$ randomly
Select the winner of each pair using the lower fitness (Ff_i)
Winner *VVk* is made a member of mating pool
Perform the above selection till mating pool is full

Step 7: Variation

Repeat for every i^{th} and $(i+1)^{th}$ $(i=1, 3, \ldots mp-1)$ *VVk* in mating pool
Perform a modified single point crossover (Davis, 1985) with a probability p_c ensuring no duplicate *Vid* in the view vector.
Perform random mutation (Goldberg, 1989) with a probability p_m ensuring no duplicate *Vid* in view vector.
Assign the resultant population to *POna*.

Repeat Step 2 to Step 7 WHILE *generation t++* <= n_g
END of Algorithm

$BDVSA_{SPEA\text{-}2}$ comprises seven steps. The algorithm uses view vectors of size *k*, which represent the number of materialized views.

In step 1, a random population (*POna*) of size *N* with view vectors (*VVk*) of size *k* is created and the archived population is set to null. The algorithm then performs steps 2 to 7 for a pre-specified number of generations.

In step 2, the values of the objective functions, i.e. query evaluation cost and update processing cost are computed. For each view vector of the population, the minimum cost of evaluating a query is computed using equation (1) and the *QEC* is computed using equation (2). These computations use the alternate query evaluation plans, the size of the views or the data required to compute the view (in

terms of Big data blocks), the change in the size of data or views, and query frequencies. Equation (3) is used to compute the *UPC*. It requires the size of data (in terms of Big data blocks) that is processed to perform updates on the materialized views and the integrity factor of the data of the views.

Step 3 is used to assign fitness to each view vector VVk_i in the population and the archived population. The strength of a view vector is computed by finding the number of view vectors it dominates. The strength value (*StVVk*) for each view in the population and the archived population is computed using equation (4) (Zitzler et al. 2001)

$$StVVk_i = \left|\{VVk_j \mid VVk_j \in \left\{PONa \cup POarc\right\} \wedge VVk_i \succ VVk_j\}\right| \tag{4}$$

The strength of the i^{th} view vector is computed (equation 4) by computing the cardinality of the set of view vectors, which are dominated by the i^{th} view vector in the set of population and archived population. An i^{th} view vector dominates a j^{th} view vector, represented by $VVk_i \succ VVk_j$, if the i^{th} view vector is better than the j^{th} view vector in at least one objective function and is as good as the j^{th} view vector with regard to the remaining objective functions. Next, the raw fitness value (Rf_i) for each VVk_i is computed using equation (Zitzler et al. 2001) given below:

$$Rf_i = \sum_{\left[\left(j \in \left\{PONa \cup POarc\right\}\right) \wedge \left(VVk_j \succ VVk_i\right)\right]} (StVVk_j) \tag{5}$$

The raw fitness of a view vector is the sum of the strength of all the view vectors in the population and the archived population that dominates it. Thus, a view vector, which is not dominated by any other view vector will have raw fitness as 0.

Next, the density value (den_i) is computed for each of the view vector VVk_i using the *k*-nearest distance from all other view vectors in the population and the archived population, which is given below (Zitzler et al. 2001).

$$den_i = \frac{1}{(\alpha^k_{VVk_i} + 2)} \tag{6}$$

The value 2 in the denominator of the above equation is a constant value, which ensures that the value of the density is always less than 1. Further, $\alpha^k_{VVk_i}$ is computed using the Euclidian distance of the view vector VVk_i from all other view vectors in the population and the archived population. The Euclidian distance is computed after normalizing the *QEC* and *UPC* values to *nQPC* and *nUPC* respectively, as the values of *QEC* and *UPC* differ in magnitude. The distance between the i^{th} and the j^{th} view vectors is computed as given below (Zitzler et al. 2001):

$$dist_{ij} = \sqrt{\left(nQEC_i - nQEC_j\right)^2 + \left(nUPC_i - nUPC_j\right)^2} \tag{7}$$

Next, the distance ($dist_{ij}$) for VVk_i are sorted and the k^{th} distance $\left(\alpha^k_{VVk_i}\right)$ is used for the computation of density (den_i) of the i^{th} view vector. The value of *k* in the context of the *k*-nearest neighbor method is computed using the expression $\sqrt{\left(N + Narc\right)}$ (Zitzler et al. 2001)

The fitness value (Ff_i) for the i^{th} view vector (VVk_i) is computed using the following equation (Zitzler et al. 2001), where raw fitness (Rf_i) is computed using equation (5) and density (den_i) is computed using equation (6):

$$Ff_i = Rf_i + \ den_i \tag{8}$$

Step 4 of the algorithm is used for the selection of the environment. For this, first, the non-dominated view vectors are identified and transferred to the archived population for the next generation $\left(POarc\right)_{t+1}$. This can be computed as given below (Zitzler et al. 2001):

$$\left(POarc\right)_{t+1} = \{VVk_i \mid \ VVk_i \in \left\{\left(PONa\right)_t \cup \left(POarc\right)_t\right\} \wedge \ Ff_i < 1\} \tag{9}$$

The above equation identifies all the VVk_i in the population or the archived population, which has the fitness value of less than 1 and creates the archived population of the next generation. It may be noted that all the non-dominated view vectors must have a fitness value less than 1, as for a non-dominated view vector, raw fitness (Rf_i) (equation 5) would be zero, and the value of the density (den_i) (equation 6) would always be less than 1, as the value of the denominator of equation (6) is atleast 2.

Next, the size of this next generation archive ($\left|\left(POarc\right)_{t+1}\right|$) is compared with the archive size (*Narc*). In case both are of the same size, no further action is required, however, if the number of non-dominated view vectors in $\left|\left(POarc\right)_{t+1}\right|$ is less than *Narc*, then [(*Narc*) - $\left|\left(POarc\right)_{t+1}\right|$], best dominated view vectors from *POna* and *POarc* are transferred to the archived population of the next generation. If the value of $\left|\left(POarc\right)_{t+1}\right|$ is more than *Narc* then the view vectors which are closest to other view vectors are truncated using the density value. In each truncation operation, only one view vector having lesser density is removed. Thus, $\left|\left(POarc\right)_{t+1}\right|$ - (*Narc*) truncation operations are performed.

In step 5, the algorithm checks, if the algorithm has run for the pre-specified number of generations. If so, the non-dominated view-vectors stored in the archived population $(POarc)_{t+1}$ are produced as the output and the algorithm terminates.

Step 6 of the algorithm uses the binary tournament selection method with replacement on $(POarc)_{t+1}$ to create the mating pool (Zitzler et al. 2001). A pair of *VVk*s are randomly selected from the $(POarc)_{t+1}$. The view vector *VVk,* which has a lower fitness value (Ff_i), is selected and added to the mating pool. This operation is repeated till the mating pool is full.

Step 7 is used to perform the crossover and mutation operations. The crossover is performed on a pair of view vectors *VVk* in the mating pool using a modified single point crossover operator (Davis, 1985) with a probability p_c. This modified single point crossover ensures that no duplicate view is part of a view vector. Similarly, random mutation (Goldberg, 1989) is performed with a probability p_m. The modified random mutation operation further ensures that no duplicate view is part of a view vector.

Steps 2 to step 7 of the algorithm are repeated for predefined number of generations. An example illustrating the use of $BDVSA_{SPEA\text{-}2}$ is discussed next.

5 EXAMPLE OF $BDVSA_{SPEA-2}$

The input to the proposed algorithm is a set of queries and their frequencies, which are shown in Table 1. These queries assume three query attributes Q_A {*A, B*, *C*} with the dependency $B \rightarrow C$, as shown in Fig. 1. This example assume a Big data size of 100 Big data blocks (one Block = 128 MB), which includes 10 Big data blocks of structured data, 40 Big data blocks of semi-structured data and 50 Big data blocks of unstructured data. Figure 1 is also used to create the candidate views as shown in Table 2, for example, the view *V1(2)* in Table 2 has been created using the second node *View(B,A)* in Fig. 1, using the structured and semi-structured data (*SD+SSD*) only. The view cost data, viz. CV_i and CMV_i, are also computed using the data size, in terms of the number of Big data blocks, using Fig. 1. For example, the CV_1 for view *V1* is computed by adding the sizes of the structured and the semi-structured data of the *Root Node* and CMV_1 for view *V1* is computed by adding the sizes of the structured and the semi-structured data of the second node *View(B,A)*. The other statistic shown in Table 2, is based on different types of the data of the views. Table 1 shows a set of 5 most frequent queries, which uses 8 possible views. It also shows the alternate query evaluation plans, which are generated using the links or interrelationships shown in Fig. 1 and the type of data used to create the view. Table 2 shows these 8 views and the data related to these views. The views 9, 10 and 11

Table 1. Query evaluation Plans and Query frequency

Query Evaluation Plans for different Queries					*Query Frequency*
Q_1	*V9*	*V5*	-	-	10
Q_2	*V10*	*V7*	*V1*	*V3*	20
Q_3	*V11*	*V8*	*V2*	*V4*	20
Q_4	*V9, V10*	*V9, V3*	-	-	30
Q_5	*V9, V11*	*V6*	*V2*	-	20

Table 2. Example data of Views for materialization

V1 (2)	*V2 (2)*	*V3 (3)*	*V4 (4)*	*V5 (5)*	*V6 (5)*	*V7 (6)*	*V8 (6)*	*V9 (Root)*	*V10 (Root)*	*V11 (Root)*
SD+SSD	*SD+UD*	*SSD*	*SD+UD*	*SD*	*SD+UD*	*SSD*	*UD*	*SD*	*SSD*	*UD*
Views Cost Data:										
Size of non-materialized view (CV_i):										
50	60	40	60	10	60	40	50	10	40	50
Size of materialized view (CMV_i):										
26	32	20	20	4	14	8	10	10	40	50
View Integrity Factor (I_{mi})										
0.9	0.8	0.9	0.8	0.99	0.8	0.9	0.8	0.99	0.9	0.8
Size Increase Factor (ufv_i and uf_i)										
0.8	0.9	0.8	0.9	0.5	0.9	0.8	0.9	0.5	0.8	1
Data size for updating a view (UMV_i)										
40	54	32	54	5	54	32	45	5	32	50

S-Structured, *SS*-Semi-structured, *U*-Unstructured, *D*-Data

in Table 2 are created from the root node of Fig. 1 for structured, semi-structured and unstructured data respectively. These views are always assumed to be materialized.

Step 1: Generate Initial Population

An initial random population of size 10 is created and is given in Table 3. The size of the archived population is assumed to be 8. However, initially the archived population set is empty.

Step 2: Compute Objective function values

The objective function values for the population is computed using equations (2) and (3) using the data given in Table 1 and Table 2. In order to compute the *QEC*, the value of MCQ_i for every i^{th} query is computed using equation (1). For example, the value of MCQ_i for the view vector $VV4_1$ and query Q_1 is computed for the two alternate paths, which consist of a single view each viz. *V9* and *V5* respectively. It may be noted that *V9* is created from the root node and is assumed to be materialized and *V5* is one of the views in the view vector $VV4_1$ and therefore, is proposed to be selected for materialization. Thus, MCQ_i for this query for the given view vector will be computed using the following equation:

$$MCQ_i = Minimum\,of\left\{1\times CMV_9\times\left(1+\frac{1}{2}\times ufv_9\right)\right\}and\left\{1\times CMV_5\times\left(1+\frac{1}{2}\times uf_5\right)\right\}$$

Likewise, the MCQ_i for all the queries will be computed for $VV4_1$. These MCQ_i values and query frequencies will be used to compute the *QEC* for $VV4_1$ using equation (2). The computation of *UPC* for $VV4_1$ will use the data size for updating a view (UMV_i) and the integrity factor (I_m) for the views of view vector $VV4_1$, viz. (*V2, V5, V6* and *V4*), and all the root node views (*V9, V10, V11*), which are assumed to be materialized. The computed values of *QEC* and *UPC* for all the view vectors are shown in Table 3.

Table 3. The Initial Population of view vectors, QEC and UPC

View ID	*Selected Views*				*QEC*	*UPC*
$VV4_1$	2	5	6	4	4211	311
$VV4_2$	3	6	4	1	2886	318
$VV4_3$	7	6	8	5	3025	267
$VV4_4$	1	6	8	2	3604	339
$VV4_5$	6	4	2	7	3390	341
$VV4_6$	4	1	5	3	4145	256
$VV4_7$	1	3	2	7	3420	286
$VV4_8$	2	1	6	5	4167	288
$VV4_9$	5	1	4	2	4341	288
$VV4_{10}$	4	2	3	7	3072	309

Step 3: Assign Fitness

The fitness values are computed based on the strength of each view vector of size 4 ($StVV4_i$) using equation (5). A dominated matrix is created for each pair of view vectors, which is used to compute $StVV4_j$. If the view vector in the i^{th} row dominates view vector of the j^{th} column, then 1 is put in that cell, else a value 0 is put in the cell. For example, the view vector $VV4_2$ dominates view vector $VV4_4$ as it has lower *QEC* and *UPC* values. Accordingly, the row of $VV4_2$ under column $VV4_4$ has a value

1. It may be noted that $StVV4_j$ is the sum of each row, while Rf_i is obtained by adding all the $StVV4_j$ of the i^{th} column, where the dominated matrix contains a value 1 (refer to Table 4). For example, raw fitness value (Rf_i) for view vector $VV4_4$ is the sum of the strengths of $VV4_2$, $VV4_3$, $VV4_7$ and $VV4_{10}$; which dominate the view vector $VV4_4$.

Table 4. Computation of Dominated Matrix, Strength and Raw Fitness

ViewID	$VV4_1$	$VV4_2$	$VV4_3$	$VV4_4$	$VV4_5$	$VV4_6$	$VV4_7$	$VV4_8$	$VV4_9$	$VV4_{10}$	$StVV4_j$
$VV4_1$	0	0	0	0	0	0	0	0	0	0	0
$VV4_2$	0	0	0	1	1	0	0	0	0	0	2
$VV4_3$	1	0	0	1	1	0	1	1	1	1	7
$VV4_4$	0	0	0	0	0	0	0	0	0	0	0
$VV4_5$	0	0	0	0	0	0	0	0	0	0	0
$VV4_6$	1	0	0	0	0	0	0	1	1	0	3
$VV4_7$	1	0	0	1	0	0	0	1	1	0	4
$VV4_8$	1	0	0	0	0	0	0	0	1	0	2
$VV4_9$	0	0	0	0	0	0	0	0	0	0	0
$VV4_{10}$	1	0	0	1	1	0	0	0	0	0	3
	$Rf_i = \sum$ *(all dominator j's* $StVV4_j$*)*										
Rf_i	19	0	0	16	12	0	7	14	16	7	

Next, the density of each view vector is computed using equations (6) and (7). To compute density, a normalized distance from the i^{th} view vector to all the view vectors is computed. These values are then sorted for each view vector. The value of k is computed as $\sqrt{(N + Narc)} = \sqrt{(10 + 8)} = 4$ *(rounded off).* Table 5 shows the distance of 4th nearest neighbor in the 4th row. Table 6 shows the density computed using equation (6).

Next, the fitness for each view vector is computed using equation (8). Table 6 shows the fitness assignment.

Table 5. Computation of k^th^ Nearest Neighbor

	$VV4_1$	$VV4_2$	$VV4_3$	$VV4_4$	$VV4_5$	$VV4_6$	$VV4_7$	$VV4_8$	$VV4_9$	$VV4_{10}$
$\alpha^1_{VV4_i}$	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
$\alpha^2_{VV4_i}$	0.8163	0.4606	0.9739	0.3974	0.3974	1.1262	0.9739	0.3159	0.3159	0.4606
$\alpha^3_{VV4_i}$	0.8460	1.2237	1.4737	1.4211	1.2237	1.1805	1.0292	0.8163	0.8460	1.0292
$\alpha^4_{VV4_i}$	**1.4826**	**1.4856**	**1.8022**	**1.4826**	**1.2650**	**1.6995**	**1.3573**	**1.1262**	**1.1805**	**1.2650**
$\alpha^5_{VV4_i}$	1.6756	1.4938	2.0757	1.4938	1.8378	1.9416	1.4856	1.3573	1.6731	1.4211
$\alpha^6_{VV4_i}$	1.8378	1.8022	2.1916	1.8845	1.9387	2.0757	1.6731	2.0735	2.2461	1.4737
$\alpha^7_{VV4_i}$	1.9416	2.4201	2.4926	2.0735	2.3562	2.7124	1.6756	2.1291	2.4268	2.0688
$\alpha^8_{VV4_i}$	2.0688	2.5623	2.6370	2.2461	2.5580	3.0898	1.6995	2.1916	2.4926	2.1291
$\alpha^9_{VV4_i}$	2.4201	2.8522	2.6800	2.7246	2.6800	3.1729	1.8845	2.3562	2.5580	2.4268
$\alpha^{10}_{VV4_i}$	2.6370	3.1729	2.7246	3.0898	3.3099	3.3099	1.9387	2.5623	2.8522	2.7124

Table 6. Computation of Final Fitness

View ID	*Selected Views*				$StVV4_i$	Rf_i	Den_i	Ff_i
$VV4_1$	2	5	6	4	0	19	0.28714	19.287
$VV4_2$	3	6	4	1	2	0	0.2869	0.2869
$VV4_3$	7	6	8	5	7	0	0.26301	0.26301
$VV4_4$	1	6	8	2	0	16	0.28714	16.287
$VV4_5$	6	4	2	7	0	12	0.30628	12.306
$VV4_6$	4	1	5	3	3	0	0.27031	0.27031
$VV4_7$	1	3	2	7	4	7	0.29786	7.2979
$VV4_8$	2	1	6	5	2	14	0.31987	14.32
$VV4_9$	5	1	4	2	0	16	0.31442	16.314
$VV4_{10}$	4	2	3	7	3	7	0.30628	7.3063

Step 4: Selection of Environment

Selection of the environment is performed using equation (9). Since, the number of non-dominated view vectors is less than the size of the archived population, the best dominated view vectors are transferred to the archived population of the next generation. Table 7 shows this archived population of the next generation.

Table 7. Archived Population of size 8 for the Next Generation

View ID	*Selected Views*				*QEC*	*UPC*	*Ff.i*
$VV4_2$	3	6	4	1	2886	318	0.2869
$VV4_3$	7	6	8	5	3025	267	0.26301
$VV4_6$	4	1	5	3	4145	256	0.27031
$VV4_7$	1	3	2	7	3420	286	7.2979
$VV4_{10}$	4	2	3	7	3072	309	7.3063
$VV4_5$	6	4	2	7	3390	341	12.306
$VV4_8$	2	1	6	5	4167	288	14.32
$VV4_4$	1	6	8	2	3604	339	16.287

Step 5: Termination

Since the generations are less than the number of specified generations, the algorithm proceeds to step 6.

Step 6: Mating Selection

For mating selection, binary tournament selection is performed on the views using the next generation archived population. The mating pool is shown in Table 8.

Step 7: Variation

The modified crossover is performed with a probability of 0.8 and mutation is performed with a probability of 0.1. The results of the variation are shown in Table 8.

Table 8. Mating Pool, Crossover and Mutation

Mating Pool				*Crossover Point*	*After Crossover*				*Mutation Point*	*After Mutation*			
4	1	5	3	1	4	6	8	5	-	4	6	8	5
7	6	8	5		7	1	5	3	1	6	1	5	8
4	1	5	3	2	4	1	3	7	-	4	1	3	7
4	2	3	7		4	2	5	3	-	4	2	5	3
4	1	5	3	3	4	1	5	6	-	4	1	5	6
7	6	8	5		7	6	8	3	-	7	6	8	3
4	1	5	3	2	4	1	5	3	-	4	1	5	3
4	1	5	3		4	1	5	3	-	4	1	5	3
6	4	2	7	3	6	4	2	7	-	6	4	2	7
1	3	2	7		1	3	2	7	-	1	3	2	7

Steps 2 to 7 are repeated for a pre-specified number of generations whereafter, the non-dominated view vectors of size k are produced as the output.

$BDVSA_{SPEA\text{-}2}$ was executed for 10 generations for the view vectors of size 4 (k=4). The non-dominated view vectors of size 4 are shown in Table 9.

Experimental results are discussed next.

Table 9. Non-Dominated View Vectors of size 4

k	*Selected Views*					*QEC*	*UPC*
4	7	6	8	5		3025	267
4	7	6	8	3		2260	298
4	7	6	5	3		3345	247
4	7	1	5	3		4679	224
4	3	6	4	5		2811	279

6 EXPERIMENTAL RESULTS

The $BDVSA_{SPEA\text{-}2}$ is implemented using GNU Octave 4.4.1 on an Intel dual core I5, 2.5 GHz, 64-bit processor, with 6 GB RAM. This implementation assumes a set of 14 queries with 4 query attributes and 16 views and similar query statistics as was considered in (Kumar A, Vijay Kumar T.V, 2021b) so as to validate the results. It may be noted that most of the statistics, which have been used for experimental results are similar to the example data which has been presented in section 5. The algorithm was run for a population (N) =100; an archived population size = 80; Number of generations (n_g) = 25; Probability of crossover (p_c) = 0.8; and Probability of mutation (p_m) = 0.1.

The algorithm was run for different sizes of the view vectors (k). Figure 3 shows the non-dominated view vectors for different sizes of the view vectors (k) from 3 to 10. It can be observed form Fig. 3 that for lower values of k, the non-dominated view vectors have a higher query evaluation cost and lower view update cost.

Figure 4 represents the *QEC, UPC* and the number of materialized views. It shows that the query evaluation cost decreases, in general, with an increase in the number of materialized views while the update processing cost increases with an increase in the number of materialized views. Figure 5 shows non-dominated view vectors of all sizes.

Figure 3. Query Evaluation Cost and Update Processing Costs for view vectors of different sizes

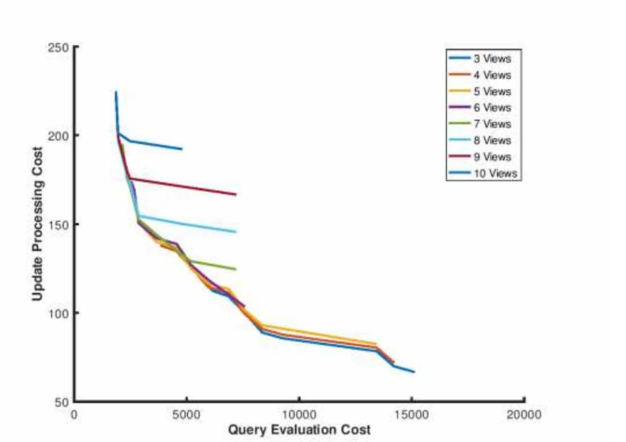


Figure 4. Query Evaluating Cost Vs. View Update cost Vs. Number of Materialized Views

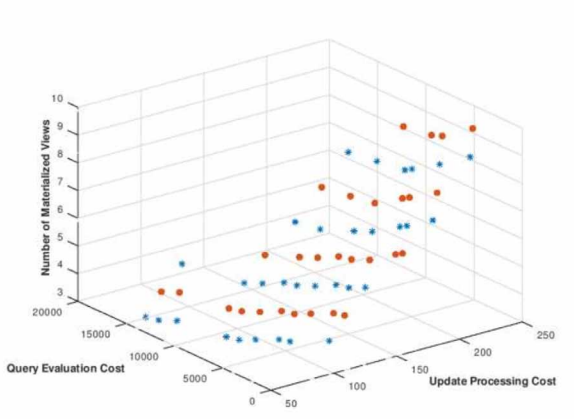
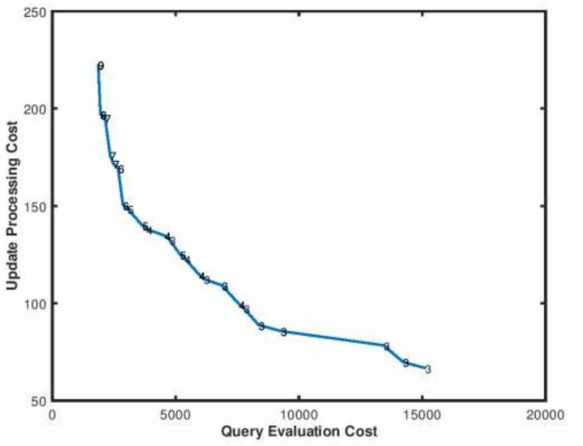


Figure 5. Non-dominated view vectors



It may be noted that only 24 distinct non-dominated data points are visible in Fig. 5. Each of these data points refer to a unique value of *QEC* and *UPC* pair. However, it was observed that dissimilar view vectors may have same *QEC* and *UPC* values (Table 10). In fact, $BDVSA_{SPEA-2}$ has produced 72 different non-dominated view vectors.

Table 10. View vectors of size 9 having the same QEC and UPC

QEC	*UPC*	*View Vectors of Size 9*
1859	223	2 4 5 7 9 10 11 12 13
1859	223	2 4 5 6 9 10 11 12 13
1859	223	2 4 5 8 9 10 11 12 13

It can be noted from Table 10 that, although the three view vectors of size 9 though have different sets of materialized views, but they have the same values of *QEC* and *UPC*. This can be attributed to the experimental data set, which has been used to compute the set of materialized views. It may be noted that the three view vectors in Table 10 differ only on one view, i.e. the fourth row.

$BDVSA_{SPEA-2}$ performs better than the greedy approach in regards to electing Big data views for materialization (Kumar & Vijay Kumar, 2021a), as the latter produces only one view vector for each view vector of size *k*.

Table 11 compares the non-dominated view vectors obtained from $BDVSA_{SPEA-2}$ with the view vectors obtained from $BDVSA_{VEGA}$ (Kumar & Vijay Kumar, 2021b). Table 11 shows view vectors of size k=7. It can be noted from table 11 that the view vectors obtained from $BDVSA_{SPEA-2}$ dominate the view vectors obtained by $BDVSA_{VEGA}$. Thus, $BDVSA_{SPEA-2}$ produces better quality view vectors in comparison of $BDVSA_{VEGA}$ (Kumar & Vijay Kumar, 2021b).

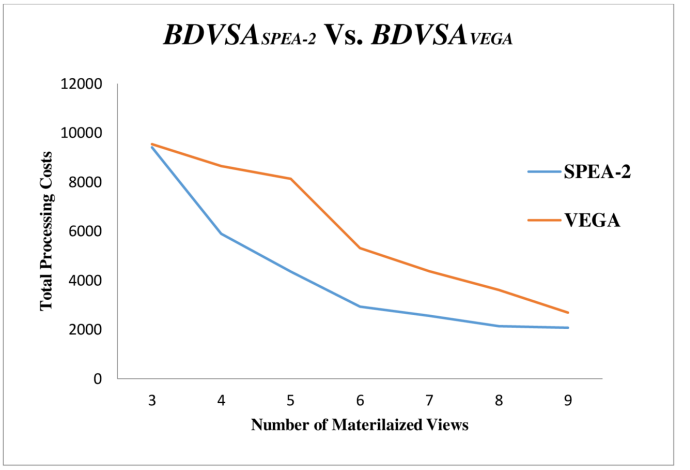
Table 11. $BDVSA_{SPEA-2}$ Vs. $BDVSA_{VEGA}$: QEC and UPC of view vectors

View Vectors of $BDVSA_{SPEA-2}$		*View Vectors of* $BDVSA_{VEGA}$	
QEC	*UPC*	*QEC*	*UPC*
2138	195	2826	210
2341	176	2956	215
2476	172	3159	196

Further, $BDVSA_{SPEA-2}$ produces a number of view vectors, any of which can be selected by an application for materializing the Big data views subject to the space and timing constraints pertaining to that application.

The total processing cost, which is computed by adding the query evaluation cost and the update processing cost, was plotted against the number of materialized views (Fig. 6). This total processing cost was averaged on the final set of view vectors produced by $BDVSA_{SPEA-2}$ (Fig. 5) and the best five view vectors on *QEC* along with the best five vectors on *UPC* of $BDVSA_{VEGA}$. However, it may also be noted that $BDVSA_{SPEA-2}$ produces non-dominated view vectors, whereas $BDVSA_{VEGA}$ does not produce such view-vectors. Thus, in general, $BDVSA_{SPEA-2}$ performs comparatively better than $BDVSA_{VEGA}$. However, these algorithms can be compared using statistical approaches.

Figure 6. $BDVSA_{SPEA-2}$ Vs. $BDVSA_{VEGA}$: Average total processing cost of view vectors



One of the sources of errors, which may lead to the non-optimal selection of Big data views for materialization, is the query frequencies statistics, which has a major impact on the computation of the query evaluation cost. It may be noted that the present statistics of query frequencies are used to estimate the future statistics of queries. In general, Big data application queries may show a periodicity over time and should be collected as part of the statistics of an application. In addition, other statistics, as shown in Table 2, also needs to be optimized for minimizing errors in view materialization. Seamless implementation of $BDVSA_{SPEA-2}$ in an actual Big data application on different master nodes may be considered, as it can minimize the query processing time.

7 CONCLUSION

Big data view materialization is formulated as a bi-objective Big data view materialization problem in (Kumar & Vijay Kumar, 2021b) where the two objectives are the minimization of the query evaluation cost for a set of workload queries and the minimization of the update processing cost of the materialized views over a given time window. This bi-objective Big data view materialization is addressed using *SPEA-2* in this paper. Accordingly, *SPEA-2* based Big data view selection algorithm, i.e. $BDVSA_{SPEA-2}$, that selects non-dominated and diverse Big data views for materialization is presented. Further, it shown that the algorithm $BDVSA_{SPEA-2}$ is able select comparatively better quality views than those selected using existing Big data view selection algorithms. Big data views selected using $BDVSA_{SPEA-2}$, when materialized, would improve the query response times of Big data queries. As a future work, the performance of $BDVSA_{SPEA-2}$ can be statistically compared with other existing Big data view selection algorithms using larger and real experimental data sets.

FUNDING AGENCY

The publisher has waived the Open Access Processing fee for this article.

REFERENCES

Abiteboul, S. (1999, December). On Views and XML. *SIGMOD Record*, *28*(4), 30–38. doi:10.1145/344816.344853

Abiteboul, S., Goldman, R., McHugh, J., Vassalos, V., & Zhuge, Y. (1997). *Views for Semistructured Data*. Technical Report. Stanford InfoLab, Workshop on Management of Semistructured Data, Tucson, AZ.

Agrawal, S., Chaudhari, S., & Narasayya, V. (2000). Automated Selection of Materialized Views and Indexes in SQL databases. *26th International Conference on Very Large Data Bases (VLDB 2000)*, 486-505.

Arun, B., & Vijay Kumar, T. V. (2015a). Materialized View Selection using Marriage in Honey Bees Optimization. *International Journal of Natural Computing Research*, *5*(3), 1–25. doi:10.4018/IJNCR.2015070101

Arun, B., & Vijay Kumar, T. V. (2015b). Materialized View Selection using Improvement Based Bee Colony Optimization. *International Journal of Software Science and Computational Intelligence*, *7*(4), 35–61. doi:10.4018/IJSSCI.2015100103

Arun, B., & Vijay Kumar, T. V. (2017a). Materialized View Selection using Artificial Bee Colony Optimization. *International Journal of Intelligent Information Technologies*, *13*(1), 26–49. doi:10.4018/IJIIT.2017010102

Arun, B., & Vijay Kumar, T. V. (2017b). Materialized View Selection using Bumble Bee Mating Optimization. *International Journal of Decision Support System Technology*, *9*(3), 1–27. doi:10.4018/IJDSST.2017070101

Bibri, S. E. (2018). The IoT for smart sustainable cities of the future: An analytical framework for sensor-based big data applications for environmental sustainability. *Sustainable Cities and Society*, *38*, 230–253. doi:10.1016/j.scs.2017.12.034

Chirkova, R., Halevy, A. Y., & Suciu, D. (2001). A Formal Perspective on the View Selection Problem. *Proceedings of the 27th VLDB Conference*.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. *Proceedings of the international joint conference on artificial intelligence*, 162–164.

Dean, J., & Ghemawat, S. (2012, January). MapReduce: A Flexible data processing tool. *Communications of the ACM*, *53*(1), 72–77. doi:10.1145/1629175.1629198

Dezyre. (2015). *Hadoop Ecosystem Components and Its Architecture from the web site dated 04 Jun 2015 with Latest Update made on December 6, 2017*. Accessed on August 08, 2019 from the website. https://www.dezyre.com/article/hadoop-ecosystem-components-and-its-architecture/114

El-Sayed, M., Rundensteiner, E. A., & Mani, M. (2006). Incremental Maintenance of Materialized XQuery Views. *Proceedings of 22nd International Conference on Data Engineering (ICDE'06)*. doi:10.1109/ICDE.2006.80

Gandomi, A., & Haider, M. (2015). Beyondthe hype: Big data concepts, methods, and analytics. *International Journal of Information Management*, *35*(2), 137–144. doi:10.1016/j.ijinfomgt.2014.10.007

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Mne Learning. Addison Wesley.

Goswami, R., Bhattacharyya, D. K., & Dutta, M. (2017, December). Materialized view selection using evolutionary algorithm for speeding up big data query processing. *Journal of Intelligent Information Systems*, *49*(3), 407–433. doi:10.1007/s10844-017-0455-6

Gupta, A., & Mumick, I. S. (1995). Maintenance of Materialized Views: Problems, Techniques, and Applications. Data Eng. Bulletin, 18(2).

Gupta, H. (1996). Selection of views to materialize in a data warehouse. In F. Afrati & P. Kolaitis (Eds.), Lecture Notes in Computer Science: Vol. 1186. *Database Theory — ICDT '97. ICDT 1997*. Springer.

Gupta, R., Gupta, H., & Mohania, M. (2012). Cloud Computing and Big Data Analytics: What is new from Database Perspective? In *Proceedings of Big Data Analytics-First International Conference*. Springer. doi:10.1007/978-3-642-35542-4_5

Hadoop. (2012). https://hadoop.apache.org/

Hadoop Documentation. (2008). https://hadoop.apache.org/docs/r0.17.0/mapred_tutorial.html

Harinarayan, V., Rajaraman, A., & Ullman, J. D. (1996). Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD international conference on Management of data (SIGMOD '96)*. ACM. doi:10.1145/233269.233333

Jacobs, A. (2009, August). The Pathologies of Big Data. *Communications of the ACM*, *52*(8), 36–44. doi:10.1145/1536616.1536632

Kumar, A., & Vijay Kumar, T. V. (2015). Big data and analytics: Issues, challenges, and opportunities. *International Journal of Data Science*, *1*(2), 118–138. doi:10.1504/IJDS.2015.072412

Kumar, A., & Vijay Kumar, T. V. (2021a). View Materialization over Big Data. *International Journal of Data Analytics*, *2*(1), 61–85. doi:10.4018/IJDA.2021010103

Kumar, A., and Vijay Kumar, T.V. (2021b). A Multi-objective approach to Big data view materialization. International Journal Knowledge and Systems Science 12(2):17-3, doi:10.4018/IJKSS.2021040102.

Kumar, S., & Vijay Kumar, T.V.(2018). A Novel Quantum Inspired Evolutionary View Selection Algorithm. *Journal Sadhana, 43*(10).

Kuno, H. A., & Rundensteiner, E. A. (1995). Materialized Object-Oriented Views in MultiView. *ACM Research Issues Data Eng. Workshop*, 78-85.

Labrinidis, A., & Roussopoulos, N. (2000). WebView materialization. *ACM SIGMOD Record, 29*(2), 367–37.

Lu, W., Wang, Y., Jiang, J., Jian, L., Shen, Y., & Wei, B. (2017). Hybrid storage architecture and efficient MapReduce processing for unstructured data. *Parallel Computing*, *69*, 63–77. doi:10.1016/j.parco.2017.08.008

Luo, J., Wu, M., Gopukumar, D., & Zhao, Y. (2016). Big Data Application in Biomedical Research and Health Care: A Literature Revie. *Biomedical Informatics Insights*, *8*, BII.S31559. Advance online publication. doi:10.4137/BII.S31559 PMID:26843812

Mami, I., & Bellahsene, Z. (2012). A Survey of View Selection Methods. SIGMOD Record, 41(1).

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., & Hung Byers, A. (2011). *Big data: The next frontier for innovation, competition, and productivity*. Report by McKinsey Global Institute.

Mistry, H., Roy, P., Sudarshan, S., & Ramamritham, K. (2001). Materialized view selection and maintenance using multi-query optimization. *Proceedings of the ACM (SIGMOD) Conference on the Management of Data*, 307-318. doi:10.1145/375663.375703

Prakash, J., & Vijay Kumar, T. V. (2019a). A Multi-objective Approach for Materialized View Selection. *International Journal of Operations Research and Information Systems*, *10*(2), 1–19. doi:10.4018/IJORIS.2019040101

Prakash, J., & Vijay Kumar, T. V. (2019b). Multi-Objective Materialized View Selection using Improved Strength Pareto Evolutionary Algorithm. *International Journal of Artificial Intelligence and Machine Learning*, *9*(2), 1–21. doi:10.4018/IJAIML.2019070101

Prakash, J., & Vijay Kumar, T. V. (2020a). Multi-Objective Materialized View Selection using MOGA. *International Journal of Systems Assurance Engineering and Management*, *11*(2), 220–231. doi:10.1007/s13198-020-00947-2

Prakash, J., & Vijay Kumar, T. V. (2020b). Multi-Objective Materialized View Selection using NSGA-II. *International Journal of Systems Assurance Engineering and Management*, *11*(5), 972–984. doi:10.1007/s13198-020-01030-6

Ross, K., Srivastava, D., & Sudarshan, S. (1996). Materialized view maintenance and integrity constraint checking: Trading space for time. *SIGMOD Intl. Conf. on Management of Data*. doi:10.1145/233269.233361

Roussopoulos, N. (1998). Materialized Views and Data Warehouses. *SIGMOD Record*, *27*(1), 21–26. doi:10.1145/273244.273253

Tang, N., Xu Yu, J., & Tang, H. (2009), Materialized View Selection in XML Databases. Database Systems for Advanced Applications, 5463.

Vijay Kumar, T. V., & Arun, B. (2016). Materialized View Selection using BCO. *International Journal of Business Information Systems*, *22*(3), 280–301.

Vijay Kumar, T. V., & Arun, B. (2017). Materialized View Selection using HBMO. *International Journal of Systems Assurance Engineering and Management, 8*(1), 379-392.

Vijay Kumar, T. V., & Kumar, S. (2014). Materialized View Selection using Differential Evolution. *International Journal of Innovative Computing and Applications*, *6*(2), 102–113. doi:10.1504/IJICA.2014.066499

Vijay Kumar, T. V., & Kumar, S. (2015). Materialized View Selection using Randomized Algorithms. *International Journal of Business Information Systems*, *19*(2), 224–240. doi:10.1504/IJBIS.2015.069432

Yafooz, W., & Abidin, Z. (2013). Managing unstructured data in relational databases. *Proceedings - 2013 IEEE Conference on Systems, Process and Control, ICSPC 2013*, 198-203.

Zhou, C., Su, F., & Pei, T. (2020). COVID-19: Challenges to GIS with Big Data. *Geography and Sustainability, 1*(1), 77-87. https://www.sciencedirect.com/science/article/pii/S2666683920300092

Zikopoulos, P. C. E. (2011). *Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data* (1st ed.). McGraw-Hill Osborne Media.

Zitzler, E., Deb, K., & Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, *8*(2), 173–195. doi:10.1162/106365600568202 PMID:10843520

Zitzler, E., Laumanns, M., & Thiele, L. (2001). *SPEA2: Improving the strength Pareto evolutionary algorithm*. TIK-report 2001. collection.ethz.ch

Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, *3*(4), 257–271. doi:10.1109/4235.797969

Akshay Kumar is pursuing his PhD at Jawharlal Nehru University, New Delhi. He did Master of Technology (M.Tech.) in Computer Science from IIT Delhi in 1988. He is employed as a faculty member at School of Computer and Information Science, IGNOU, New Delhi.

T. V. Vijay Kumar has completed his PhD in the area of databases from Jawaharlal Nehru University, New Delhi, India, after completing his MPhil and MSc in Operational Research, and BSc (Hons) in mathematics, from the University of Delhi, Delhi, India. His research interests are databases, data warehousing, data mining, machine learning, nature inspired algorithms, disaster management, Big Data, and analytics.