Chapter 1

Introduction

We are drowning in an ocean of data. Every minute Email users send over 200 million messages, Twitter users tweet nearly 300,000 times, YouTube users upload 72 hours of new video content, Instagram users post nearly 220,000 new photos and Amazon generates over $80,000 in online sales[1]. Facebook stores more than 300 petabyte of data in their Hive technology based data warehouse with an incoming rate of around 600 terabyte of data every minute which is increasing three times every year[2]. During the month of August 2015, there were 1.49 billion daily users of Facebook[3].

Historical data are manipulated for deriving data for analysis by applying analytical operations such as ratios, totals, trends, allocations across dimensions and across hierarchical levels to plan and forecast for better productivity and earnings by defining gaps and deciding on new strategies. These analytical processing do not require any transaction processing. Therefore these historical data are kept separately in *data warehouses*. Inmon in [1] defines data warehouse as Definition 1 below.

Definition 1. *A **data warehouse** is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision making process.*

The results of some of the very expensive analytical operations on data warehouse may be saved for using them by other similar subsequent analytical processing on these huge volume of data. Therefore, data warehouse views are used to pre-compute and store aggregated data of a central theme such as the sum of sales, grouped by different classifications of the theme. For example, total sales from January to June in a year, total quarterly sales in a particular region of a country in a specific year, total sales by a dealer etc.. The views are often referred to as summaries in data warehouses, because they are used to store sum-

[1]Guneltus, S. in: The data explosion in 2014 minute by minute infographic, 2014. URL http://aci.info/

[2]As posted in "https://code.facebook.com" in October 2015

[3]The US edition of The Times of India, September 13, 2015.

marized data. They are also used to pre-compute database joins with or without aggregations.

Definition 2. *In database theory, a **view** is the result set of a stored query on the data, which the database users can query just as they would in a persistent database collection object.*

Materialized or stored views as defined in Definition 3 below are primarily used to eliminate the overhead associated with expensive joins and aggregations for a number of selected queries when they are executed again and again.

Definition 3. *In computing, a **materialized view** is a database object that contains the results of a query or a portion of a query.*

Materialized views are also can be used to replicate data at distributed sites and to synchronize updates at distributed sites with conflict resolution methods for distributed computing. Again materialized views may be used as replicas to provide local access to data that otherwise would have to be accessed from remote sites. In mobile computing, materialized views can be used to download a subset of data from central servers to mobile clients with periodic updates between the mobile clients and the central servers.

Large amounts of storage space are required for materializing the views. Materializing view also incurs maintenance costs for periodic refreshing and updates. Therefore it is not feasible to save all views of a data warehouse. And hence, an optimum set of views are selected for materializing.

In this thesis an analysis of applying few randomized search methods, customized for selecting views for materializing, is presented.

This chapter focuses on introducing the use of materialized views in data warehouse and the materialized view selection problem as multi-objective optimization problem. The objectives of this research work, contributions and organization of this thesis are also presented in this chapter.

1.1 Data Warehousing and Materialized Views

Data warehouse consolidates data in multidimensional space by integrating data from multiple heterogeneous data sources. Data warehouses are used for interactive analysis of multidimensional data of different levels of granularity for making strategic decisions. They are maintained separately from operational database as they are used only for analysis for using as support system for management's decision making process and do not require any transaction processing. The basic operations of data analysis on data warehouse are analytical processing by On Line Analytical Processing (OLAP) system. Thus Data warehouses are historical operational data for analytical processing and data mining [2–4]. In case of operational database, basic operation is transactional processing. Therefore data in

operational database may change for every transactional operation performed on them. While making strategic decision by analytical processing, the database is considered to be unchanged during the considered period. Therefore, from multiple transactional databases data are uploaded periodically to a data warehouse for analytical processing later on. For analytical processing, aggregated or consolidated measures of some selected entities of the transactional information system over a large historical period are useful which are not very important in case of transactional database. Therefore the data warehouses are maintained separately from transactional database to maintain the performance of both the systems. Data warehouses are designed considering a central theme of measure on different related entities. The entities that are measured and used by OLAP functions for analyzing the central theme are termed as dimensions of the data warehouse. The number of enquiries may be the central theme of a data warehouse for Example 1 below.

Example 1. *The Indian Railways run around 7000 passenger trains daily. There are 18 different types of trains or services for passengers with 10 different classes of services in them connecting 7112 stations managed by 17 different administrative zones. Indian Railway carries around 13 million passengers every day. Most of the passengers make enquiry regarding availability of different services before booking a service*[4].

In Example 1, the enquiries are made on real time transactional database to get the latest updated information. On the other hand for strategic planning, the users needs may be analyzed by studying different enquiries triggered to the system without any transactional updating. All the queries made over a time period to the system may be recorded and loaded to a separate data warehouse. For strategic decisions, different OLAP operations are to be performed on this data warehouse of enquiries made regarding different services or entities at different times of the organization. Hence, aggregated values on number of queries on different dimensions or entities with different granularity may be recorded in the data warehouse for quicker response of OLAP queries.

1.1.1 Multidimensional data model

Data warehouse and OLAP functions are conceptualized based on a multidimensional data model. This multidimensional model is termed as *data cube*. The data cube concept visualizes data modeled in multiple *dimensions*. These dimensions are the entities, of which measures are to be recorded for analysis [2–4]. For our Example 1, we may design a data warehouse of central theme *enquiry* to keep number of queries made with respect to the dimensions zones, types of trains and time. In Relational Database Management System (RDBMS) based data warehouse, for each dimension, a table may be associated. These tables are called dimension tables. Numerical measures on different dimensions on the main subject of interest or central theme are called the *facts* of the data warehouse and

[4]http://www.indianrailways.gov.in/

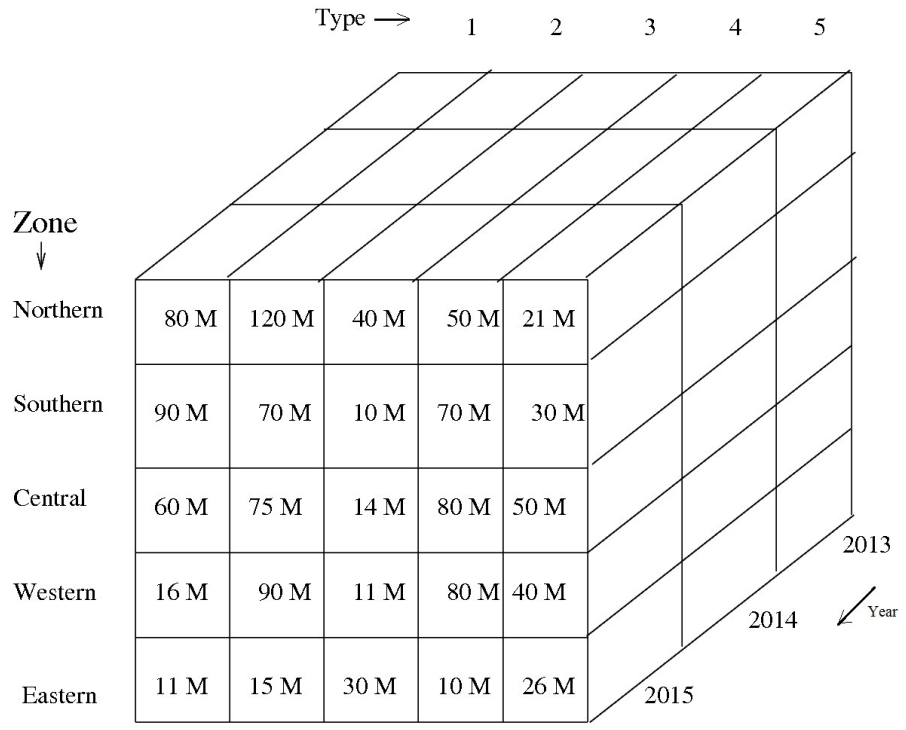

Figure 1-1: Data cube representing a data warehouse.

are represented by a central *fact table*. In case of Example 1, number of queries made with respect to different zones, stations, routes, types of services and time of inquiries are the facts that are to be recorded in a fact table. The data cube representing the data warehouse in Example 1 may be represented as Figure 1-1. A cube in geometry is a 3-D geometric structure, however in case of data warehousing, the data cube is of n-dimensional for n number of dimensions on which facts or measures are recorded. As depicted in Figure 1-1, *cuboid*s for each of the subsets of given dimensions may be generated. Different degrees of summarized measures or data are stored in these cuboids. Thus, *lattice* of cuboids providing data at a different level of aggregation or "group by" is formed. These lattices of cuboids are termed as data cubes. Figure 1-2 represents a lattice of cuboids as a data cube for dimensions type, zone and year. The cuboid keeping lowest level of summarized value is called the base cuboid and cuboid containing highest level of summarisation is called the apex cuboid. The value of a measure in a data cube is a numerical function evaluated for a given point by aggregating the data corresponding to the specific dimension and value pairs. Some example of aggregating functions are sum(), min(), max() and count(). Again a dimension may have several levels of concept hierarchies, e.g. the time dimension based on the attributes day, week, month, quarter and year on which aggregated values are to be kept (see Figure 1-3). For implementing data cube concept of data warehousing in entity-relationship data model, three types of modeling paradigm called star-schema, snow-flake schema and fast constellation schema are used [2,4]. These schema basically contain a central fact table containing the primary-keys of other dimension tables as foreign key and related measures or values. The dimension tables contain a primary key field and the hierarchical data. For ex-

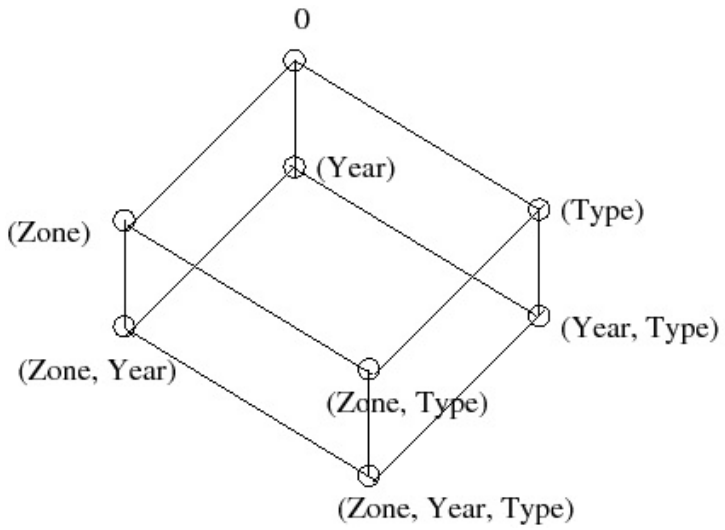

Figure 1-2: A lattice of cuboids

ample the fields in a fact table may be time_key, service_type_key, location_key and number_of_inquiries. Similarly dimension table time may contain fields like time_key, time_of_day, day, day_of_the_week, month, quarter, year. Mainly four types of OLAP operations are performed on multidimensional data model based data warehouses. The OLAP operations are termed as *Roll-up, Drill-down, Slice and dice* and *pivot (rotate)* [2]. The roll-up is performed for aggregating values reducing one or more dimensions from the data cube like representing number of queries about a service in a period of time by summing up values of all different zones where the zone dimension is removed. The Drill-down is the reverse operation of roll-up. That is, by drill-down, data in a new dimension are added by evaluating and exploding the aggregated values in the new dimension. The slice operation is selection of one dimensional data of a cube and the dice operation is for generating a sub-cube by performing a selection on two or more dimensions. The pivot is a data visualization operation by rotating the data axes to present an alternative view of the data in the data cube.

1.1.2 Aggregations of data as data warehouse views

The main design consideration of data cube based multidimensional data warehouse is efficient computation of aggregations of data across different dimensions. In SQL implementation of multidimensional data warehouse, the aggregations across dimensions are performed by group-by clause of SQL. Thus a cuboid is represented by an SQL group-by clause. Thus a set of SQL group-by forms a lattice of cuboids or data cube [2–4]. An example SQL implementation of data cube is presented below.

Example 2. *SQL implementation of a data cube*

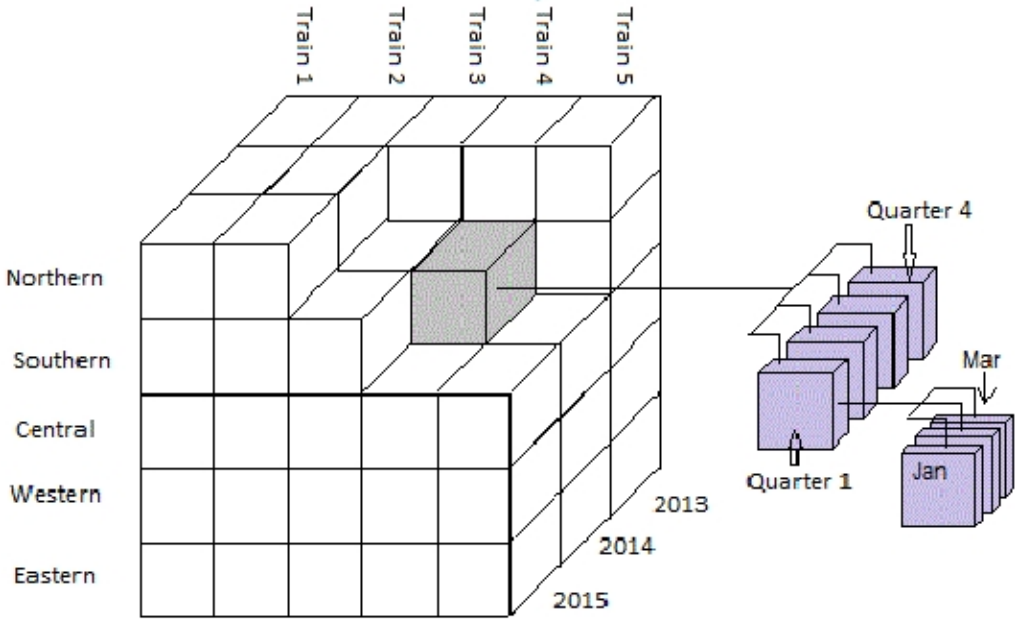

Figure 1-3: Representation of hierarchies in Data cube

```
SELECT s.time_key, s.service_type_key, s.location_in_zone_key,
SUM(s.numb_of_enquiry)
FROM enqiry s, time m, service_type t, zone z,
WHERE s.time_key=m.time_key
AND s.service_type_key=t.service_type_key
AND s.location_in_zone_key=z.location_in_zone_key
GROUP BY s.time_key, s.service_type_key, s.location_in_zone_key;
```

For three dimensions there may 2^3=8 number of possible "group-by"s like (zone, type, year), (zone, type), (zone, year), (type, year), (zone), (type), (year), (). Thus for n-dimensional data cube, there will be total 2^n number of cuboids. Again many of the dimensions have hierarchies. So, for n-dimensional data cube, $\Pi_{i=1}^{n}(L_i+1)$ number of cuboids can be generated, where L_i represents the number of levels or hierarchies associated with a dimension i. (L_i+1) is used to include the top level i.e group-by all [2]. Analytical processing on data warehouse accesses different cuboids for responding decision support queries. Therefore, all or at least some of the cuboids in a data cube may be computed in advance during an analytical processing session to make query response faster. These pre-computed aggregations are termed as data warehouse **views**.

1.1.3 Materialized views for efficient computation of data cubes

Some data warehouse views may be generated frequently in different analytical processing sessions by some frequent queries in a period of time in the past on the data warehouse. Therefore these views may be saved or stored in the data warehouse as *materialized views* to avoid redundant computations while responding OLAP queries. Thus materialized views make query response significantly faster. For large number of dimensions, conceptual hierarchies in dimensions, and their cardinality, the storage space requirement for materializing these views may exceed the size of the data warehouse as many of the group-by's size may exceed the size of the input relation or base table [2,4]. Again, the data warehouses are to be periodically updated with corresponding operational database. Whenever there is a change in the data warehouse, the materialized views are also to be updated [3]. Therefore it is not realistic to materialize all of the views [3,4]. The computation cost of aggregations while responding OLAP queries may be substantially reduced by accessing materialized views. But substantial overhead for materializing these views do not encourage materialized views. In [4], a greedy algorithm referred as *HRU Greedy algorithm* has been proposed to determine which cuboids should be precomputed for materializing.

1.1.4 Materializing views in RDBMS technology

Most of the popular RDBMS technologies have incorporated the materialized view feature as a component in their data warehousing platforms, where SQL query can be used to materialize view for improving query response time. In RDBMS based data warehousing environment, the schema could be a star-schema, snow-flake schema or fast constellation schema without any restriction on which schema is to be used. Before creating a materialized view, the first step is to review the schema and identify the dimensions. A dimension here is an object which defines a hierarchical relationships between columns, where all the columns do not have to come from the same table [5]. Example 3 below defines a time dimension, which contains a hierarchy defining the relationship between a day, month, quarter and year in Oracle Database 10g Release 2 for a Snowflake schema of a data warehouse.

Example 3. *An SQL statement in Oracle Database 10g Release 2 to create Time dimension*

```
CREATE DIMENSION times_dim
LEVEL day IS TIMES.TIME_ID
LEVEL month IS TIMES.CALENDAR_MONTH_DESC
LEVEL quarter IS TIMES.CALENDAR_QUARTER_DESC
LEVEL year IS TIMES.CALENDAR_YEAR
HIERARCHY cal_rollup
(day CHILD OF month CHILD OF quarter CHILD OF year)
ATTRIBUTE day DETERMINES
```

```
(day_number_in_week, day_name, day_number_in_month,
calendar_week_number)
ATTRIBUTE month DETERMINES
(calendar_month_desc, calendar_month_number, calendar_month_name,
days_in_cal_month, end_of_cal_month)
ATTRIBUTE quarter DETERMINES
(calendar_quarter_desc, calendar_quarter_number, days_in_cal_quarter,
end_of_cal_quarter)
ATTRIBUTE year DETERMINES
(calendar_year, days_in_cal_year, end_of_cal_year);
```

Once the dimensions have been defined, the materialized views can be created using SQL statements. A materialized view definition in Oracle Database 10g Release 2 can include aggregation, such as SUM, MIN, MAX, AVG, COUNT(*), COUNT(x), COUNT(DISTINCT), VARIANCE, STDDEV, and a GROUP BY clause of SQL with one or more tables joined together.These views may be indexed and partitioned where basic DDL operations like CREATE, ALTER, and DROP may also be applied [5]. A materialized view is created using the CREATE MATERIALIZED VIEW statement of SQL. Example 4 below illustrates the creation of a materialized view called number_of_enquiries_mv that computes the sum of enquiries by time and service_ type_name. The SELECT list in CREATE MATERIALIZED VIEW statement must contain the entire GROUP BY columns.

Example 4. *SQL Statement to create Materialized View*

```
CREATE MATERIALIZED VIEW  number_of_enquiries_mv
PCTFREE 0 STORAGE (initial 8k next 8k pctincrease 0)
BUILD IMMEDIATE
REFRESH  FAST ON DEMAND ENABLE QUERY REWRITE
AS
SELECT time_id, service_type_name, SUM(no_of_enquiries)
AS sum_of_enquiries, COUNT(no_of_enquiries) AS
record_count_of_enquiries,
COUNT(*)  AS cnt
FROM enquiries c, types_of_services p
WHERE c.service_id = p.service_id
GROUP BY time_id, service_type_name;
```

1.1.5 Common SQL sub-expressions of SQL queries as materialized views

It has been observed that, in one hand, no materialization of pre-computed cuboids leads to expensive computing of multidimensional aggregations, and on the other

hand, pre-computing all the cuboids and materializing requires huge memory space and to incur maintenance costs. Therefore, optimum selective materialization is suggested. But clear guidelines for determining which views are to be materialized is not defined. In basic multidimensional data model of data warehousing, computing an *iceberg cube* was suggested, which is a data cube storing only those cube cells whose aggregate value is above some minimum support threshold [2]. In [4,6] authors present greedy algorithms referred as *HRU Greedy algorithm* and *Polynomial Greedy Algorithm* (PGA) to determine which cuboids should be pre-computed for materializing.

The algorithms presented in [4, 6] are to select views to materialize considering only queries with aggregate functions involved for OLAP applications but not to deal with general SQL queries that include select, project, join and aggregation operations. Most of the recent works on selecting views for materializing consider common sub-expressions among multiple queries executed frequently in analytical processing sessions or periods on data warehouse for efficient multiple query execution [7–14]. This approach of considering shared common sub-expressions of queries on data warehouse as views for materializing is thus basically outcome of the problem of selecting multiple query optimization plan [7]. Multiple-Query optimization means finding an optimal execution plan for multiple queries executed by sharing some temporary results from some common sub expressions so that the total query processing cost of all the queries are minimum. But multiple-query execution cost by sharing results of intermediate sub-expressions as views depend on how the base relations are defined and how the intermediate select, join and project operations are defined. In case of very less number of base tables an optimum query execution plan by sharing multiple number of sub-expressions or intermediate relations as views may be easier but in case of larger number of considered queries on large number of base table or relations it becomes a complex problem of deciding how the intermediate results are to be shared for minimizing the total cost of execution of the considered queries. In [7], the authors design an algorithm for generating multiple view processing plan (MVPP) for optimizing total query processing cost of multiple queries by connecting or sharing pre-identified common sub-expressions of the queries as views. By SQL queries as presented in Example 5 and 6, how sharing of result of intermediate sub-expressions of queries as materialized views can reduce total query execution cost can be shown. These two SQL queries are designed for version H benchmark database of Transaction Processing Performance Council [15].

Example 5. *A query on TPC-H benchmark database*

```
SELECT s_acctbal, s_name, n_name, p_partkey,
p_mfgr, s_address, s_phone, s_comment
FROM  part, supplier, partsupp, nation, region
WHERE p_partkey=ps_partkey AND s_suppkey=ps_suppkey
AND p_size=:1 AND p_type like '%:2'
AND s_nationkey=n_nationkey
```

```
AND n_regionkey=r_regionkey AND r_name='MIDDLE EAST'
AND  ps_supplycost=(
SELECT MIN(ps_supplycost) FROM partsupp, supplier, nation,
region WHERE p_partkey=ps_partkey AND s_suppkey=ps_suppkey AND
s_nationkey=n_nationkey AND n_regionkey=r_regionkey AND
r_name='MIDDLE EAST')
ORDER BY s_acctbal, n_name, s_name, p_partkey;
```

Example 6. *A query on TPC-H benchmark database*

```
SELECT s_acctbal, s_name, n_name, p_partkey,
p_mfgr, s_address, s_phone, s_comment
FROM  part, supplier, partsupp, nation
WHERE  p_partkey=ps_partkey AND s_suppkey=ps_suppkey
AND p_size=:1 AND  p_type like '%:2'
AND s_nationkey=n_nationkey
AND n_name=':1'
ORDER BY  s_acctbal, s_name, p_partkey;
```

Though both the queries in Example 5 and 6 use different join conditions, both of them share some common sub expressions like accessing part table of 26260520 number of rows for selecting parts of "p_size=':1'" and "p_type like '%:2'" returning 26260 number of rows. If in a considered period, the frequency of execution of query in Example 5 is 20 and that of Example 6 is 10 then total $20 \times 26260520 + 10 \times 26260520$ number of rows to be accessed. By materializing or saving the intermediate result of the SELECT operation for "p_size=':1'" and "p_type like '%:2'" from the part table, these queries can be designed to access only $20 \times 26260 + 10 \times 26260$ number of rows. If number of rows to be processed or accessed is considered as query processing cost, then it is observed that there is a large amount of savings in terms of query processing cost by materializing the result of this SELECT operation as a database table called view. But during this period, the saved results or materialized views may have to be refreshed or updated. Generally the updating frequencies are much less than that of accessing the materialized views for processing the queries. During the considered period of these queries, let the materialized views are to be updated two times. Then 2×26260520 rows to be processed for maintaining the materialized view. Again for saving or materializing the intermediate results, additional space for storing 26260 numbers of rows will be required. In case of very huge data warehouse like in Example 1, there may be a large number of OLAP operations that are to be performed. In these cases, out of the large number of complex queries, some queries may be triggered very frequently. These frequent complex queries again may share a large number of sub-expressions. Therefore results of a set of sub-expressions may be materialized for minimizing the total query processing cost with minimized materialized view maintenance cost and space cost.

But, the derived relations and base relations for select, project and join operations in relational model based query processing are to be designed considering the indices and keys used in the relations. Therefore grossly selecting shared

temporary results of sub-expressions may prove to be a bad decision when indices on base relations are defined [7].

In [7], a directed acyclic graph (DAG) of multiple view processing plan (MVPP) by connecting the base relations and the selections, joins and projections considering the respective keys and indices are proposed for saving results of the derived relations as materialized views.

1.1.6 Materializing results of common sub-expressions in Big data framework

For managing very huge volume of incongruent data (in terabytes , zetta (10^{21}) bytes etc.) at right speed for analyzing in right time frame, a system model called *Big data* has been evolved. In Big data a computing paradigm called MapReduce is introduced [16]. In MapReduce model, computation tasks on Big data are to be 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 (see Figure 1-4). This system was designed for handling very big and semi structured data in a single table. A *Distributed File System* (DFS) computing framework has been designed based on the MapReduce model [17]. This DFS is designed 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 because in this DFS, the block size is at least 64MB and smaller file size of less than the block size imposes unnecessary overhead.

Hive is a framework for data warehousing on top of Hadoop DFS (HDFS) [17] of Apache with a query language called *HiveQL* similar to SQL, which was initially designed for OLAP like operations on huge volume of data that Facebook stored in HDFS [17]. Hive runs on workstations and convert HiveQL query into a series of MapReduce jobs. Hive organizes data into tables as a mean for providing structure to data stored in HDFS. But Hive is integrated with another platform called *HBase* which supports row-level update like delete, update etc. on big column oriented table of key-value type storage. Therefore, Hive supports collection data type columns STRUCT, MAP and ARRAY. Using collection data type may break normal form. In Big data systems a benefit of sacrificing normal form is higher processing throughput [18]. Again Hive has very limited indexing capabilities. Therefore, in Big data management, for analytical processing queries, the use of indices and keys in the relations are very limited. And hence, common or shared sub-expressions' results of frequent queries may be saved as materialized views. Materialized view is not currently supported by Hive. Research work is going on for supporting materialized views in HDFS [19, 20].

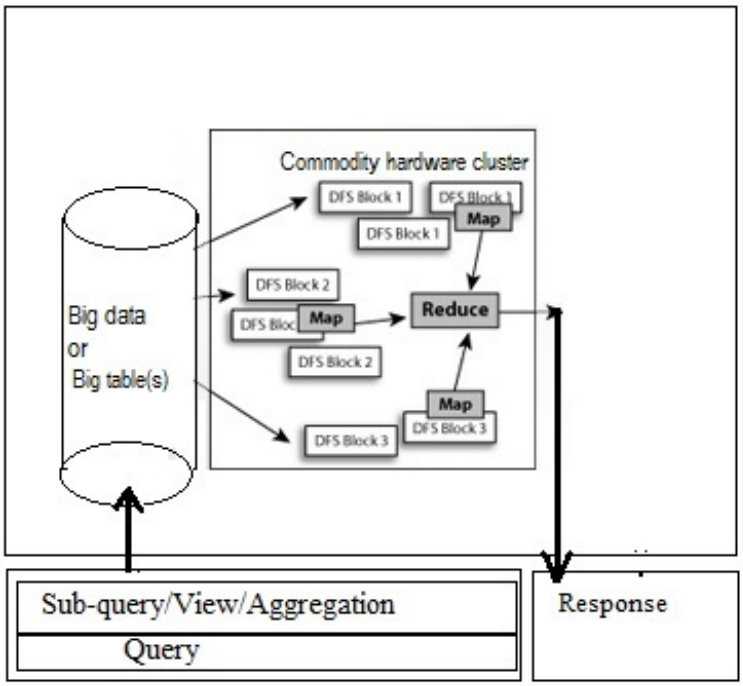

Figure 1-4: Analytical processing in MapReduce framework.

1.2 Multi-Objective Optimization to Select Views for Materializing

View materialization for reducing query processing costs in data warehouse applications requires a large amount of space. Again, materialized views are to be updated or maintained in response to changes in the base data periodically. On the other hand, not materializing any view requires lots of redundant on-the-fly computations. Therefore, it is necessary to select an optimum set of views to materialize to increase analytical query processing performance with optimized view maintenance cost and space for materializing the views.

1.2.1 The view selection problem for materializing

The materialized view selection problem is stated as- given a set of data warehouse queries, select a set of views to materialize so that the total query processing cost, materialized view maintenance cost and storage space for materializing the views are minimized [4, 21, 22]. Formally the problem may be defined as Definition 4.

Definition 4. *Given a set of n frequent queries* $Q = \{q_1, q_2, \cdots, q_n\}$ *on a data warehouse, and the set of m views* $V = \{v_1, v_2, \cdots, v_m\}$ *generated while responding the queries* Q*, a set of views* $M \subseteq V$ *are to be selected for materializing, such*

that, if A_M is the space requirement for materializing M, C_M^Q is the total cost of responding queries Q when M is materialized and $U(M)$ is the maintenance cost of materialized views M, then the selection M minimizes C_M^Q, $U(M)$ and A_M.

1.2.2 The costs to be minimized and trade-offs

The costs stated in Section 1.2.1 that are to be minimized and their dependencies are as follows:

- The maintenance or updating cost of each of the views of the set selected for materializing depends on underlying relations or base relations or tables from which that particular view is derived and the updating frequency of corresponding base relations.
- The total analytical query processing cost on the warehouse that can be minimized by materializing a set of views depends on aggregation functions, database functions like select, project, (expensive) join and "group by" on corresponding underlying base relations that are used to construct each of the selected views and frequencies of the analytical processing data warehouse queries that access these views.
- Thirdly, the total space requirement for materializing depends on individual size of the views that are selected but not on cardinality of the set of views materialized.

Thus the costs that are to be minimized depend on some underlying or internal factors but independent of each other. But there are **trade-offs** to be made because materializing more number of views may reduce the total query processing cost but there may be increase in maintenance cost and space cost of materializing the views. Also reducing number of materialized views may decrease materialized view maintenance cost and materializing space cost but it may increase the total analytical query processing cost of the data warehouse. Thus when one minimization objective improves others may degrade. Therefore, simultaneous minimization of all the costs defined in Definition 4 is not possible and trade-offs between them are to be decided.

1.2.3 Multi-Objective Optimization (MOO)

Multi-Objective optimization (MOO) is simultaneous optimization of more than one objective functions where optimal decisions are to be made considering the presence of trade-offs between objective functions, i.e, when increase in one objective function value decreases the objective function values of at least one of the other objective functions of the problem. Here trade-off means the balancing factors between the objective function values that can not be simultaneously minimized or maximized.

Formally in mathematical terms multi-objective optimization for minimization problem can be formulated as below [23].

$$minimize(f_1(x), f_2(x), \cdots, f_{\mathbf{M}}(x)),\ such\ that,\ x \in \mathbf{S} \tag{1.1}$$

where $\mathbf{M} \geq 2$ is the number of objectives and $\mathbf{S}$ is the set of all feasible solution vectors.

Multi-objective optimization is defined for maximization or minimization of objective function values where there does not exist typically a single solution that maximizes or minimizes all the objective functions simultaneously. Therefore, the multi-objective optimization technique finds the solutions of the problem that can't be farther improved for any of the objectives without farther degrading at least one of the other objective function values. In the terminology of MOO, these solutions are called *non-dominated solutions*. But by simply generating all the non-dominated solutions without indicating their distribution in objective function space it may not be possible to find the most applicable solutions or solution vectors by decision makers. Therefore, the non-dominated solutions are presented or expressed as curve (in case of two objectives) or surface (in case of more than two objectives) in objective function space known as **Pareto front** or **Pareto optimal front** such that decision maker can easily decide on trade-offs to be considered in objective function space.

1.2.4 Selecting views by multi-objective optimization technique

The view selection problem for materializing in data warehouses may be solved by converting it into a single objective optimization problem of minimizing the summed up cost function values of all the objectives functions C_M^Q, $U(M)$ and A_M for a set of materialized views M as defined in Definition 4 with some associated constraints on the costs. It returns many solution sets of views corresponding to the minimum value of summed-up cost function values. For selecting solution set of views instead of defining too many constraints on each and every objective function by the decision makers before hand, trade-offs may be decided between C_M^Q, $U(M)$ and A_M. As there are trade-offs to be considered and simultaneous minimization of all the costs defined in Definition 4 is not possible, instead of converting the problem to a single objective optimization problem by summing-up all the associated costs, the problem can be defined as a multi-objective optimization problem for solving by multi-objective optimization techniques.

Research on this problem started in the early nineties when several heuristic greedy algorithms were proposed [4, 6, 21, 22, 24]. As the problem is found to be NP-hard [4, 7, 9, 21, 22, 25], various stochastic or evolutionary algorithms and clustering based approaches have been proposed.

1.3 Motivation of this Research

In Section 1.2, it is shown that the view selection for materialization in data warehouses is basically an optimization problem with multiple objectives. But so far, in most of the works it is observed that, the problem is addressed by converting it into a single objective optimization problem. When an optimization problem with multiple non-dominating objectives is converted into single objective, it ignores that different solutions may offer trade-offs between the objectives. Though recently few attempts have been made to handle the problem by using multiple objective optimization techniques [26], yet there is scope of using other multiple objective optimization techniques in this area.

In last two decades a number of multi-objective evolutionary algorithms (MOEAs) have been developed. In [27], it is shown that Differential Evolution (DE) algorithm [28] can achieve better results than Genetic Algorithms (GAs) on numerical multi-objective optimization problems. Again non-dominated sorting genetic algorithm-II (NSGA-II) [29] was found to be able to maintain a better spread of solutions and converge better compared to other elitist MOEAs. The DE is a powerful stochastic optimization algorithm for real parameter optimization [28]. A support system may be designed for selecting views for materializing by applying evolutionary algorithms like DE, NSGA-II [29] or Multi-objective Simulated Annealing (MOSA) [30].

For managing large amount of analytic workloads on Big data systems, research works have been started for materializing frequent queries in clusters of disks, solid-state drives (SSD) and other memories [19, 20]. To cope up with the changing paradigm of very large data processing, the materialized view selection problem have to be designed in the context of distributed computations in Big data framework and to evaluate the performance of different techniques for selecting frequent sub-queries of frequent queries as views to materialize for minimizing the costs of analytical query processing.

This research is largely motivated by the observation that for strategic decision making by efficient analytical processing of historical data in data warehouses, some most relevant intermediate views with minimum associated costs are to be selected for materializing in data warehouses by a suitable optimization technique.

1.4 Research Objectives

The broad objectives of this research work are to (i) design a system for recommending a set of views or frequent sub-queries for storing as materialized views in data warehouse, considering the associated costs of maintaining the materialized views and space, and savings in total query processing costs, (ii) to find a suitable optimization algorithm for selecting the views by studying the existing works on this problem and finding the associated issues, (iii) to design input and

input methodology for the recommendation system and finally (iv) to develop a comprehensive materialized view selection mechanism compliant to the changing data warehousing and query processing paradigm.

The basic objectives of the research work are therefore may be listed as follows :

- To study different view materialization and materialized view selection techniques used in conventional data warehousing
  - To identify the associated issues and challenges involved in the problem.
  - To identify possible pitfalls and advantages in different existing representations and formulation of the problem and solutions.
  - To find the advantages and limitations of different techniques that have been so far incorporated in materialized view selection.
  - To identify the implications to theory and practices.
- Defining the view selection for materializing problem
  - By formulating and modeling a suitable representation to address the issues of existing approaches.
  - To improve the quality of solutions and performances by incorporating new or modified notions or representation of the problem and technique adaptable to the new definition.
  - To design a test bed for evaluating the performances of different view selection techniques.
- Designing a view selection system compliant to evolving Big data framework
  - By defining the problem of selection of sub-queries and aggregations for materializing in data warehouse.
  - To select a suitable optimization technique for customizing it for the problem.
  - To implement the optimization technique customized for view selection.
  - To perform experimentation for evaluating the proposed system in an accepted framework.
- Evaluating different state of the art optimization techniques for implementing in recommendation system for selecting views
  - By implementing modern and established stochastic and evolutionary optimization algorithms.
  - By measuring convergence to the optimum solutions by different techniques.
  - By designing test data for experimenting with suggested Big data paradigm and applicable algorithms.
  - By designing a prototype framework of recommendation system to provided optimum set of views for materializing.

1.5 Contributions

In this research, the problem representations, data structures and algorithms in different models proposed so far in view selection for materialization in data warehousing have been surveyed and the associated issues and challenges in addressing this NP-hard problem have been identified and reported. The view selection problem for materializing in data warehouse is defined as a multi-objective optimization problem using a cost model representing query processing plan of a set of frequent queries represented by Directed Acyclic Graph (DAG) for conventional RDBMS based data warehousing. The view selection to materialize also has been defined for Big data based Distributed File System (DFS) framework. The Multi-objective Differential Evolution (DE) algorithm has been customized for binary encoded solution representation using *Formae analysis* [31] for view selection in case of conventional RDBMS based data warehousing as well as Big data/DFS based framework. The Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [32] has also been applied using the same test-bed and test-data for comparing performances between the algorithms in materialized view selection. Finally a version of Archived Multi-Objective Simulated Annealing (AMOSA) [30] has been implemented on this problem and its performance is analyzed with other MOEAs based on *purity* and *convergence* towards Pareto front by solutions obtained and the *minimal spacing* between the solutions.

In this dissertation, the works done for achieving the above mentioned goals and contributions are discussed. The organization of the dissertation is presented in the next Section 1.6.

1.6 Organization of the Thesis

The rest of the dissertation is organized as follows.

- **Chapter 2**: *Approaches and Issues in Materialized View Selection of Data Warehouses* - In this chapter a comprehensive survey of approaches introduced to address the materialized view selection problem have been discussed. The key issues and research challenges in handling the problem are identified here with discussion on implications to related theory and practices. A comparative analysis of different approaches also have been presented based on theoretical analysis and empirical results presented in related literature.
- **Chapter 3**: *Multi-Objective Differential Evolution Algorithm for Selecting Views to Materialize* - In this chapter the view selection process for materializing in data warehouse is defined as a multi-objective optimization problem. This chapter presents an implementation of Multi-objective Differential Evolution (MODE) algorithm for binary encoded data to select views for materializing.

- **Chapter 4**: *Materialized View Selection by Evolutionary Algorithm for Big Data Query Processing* - The view selection to materialize for speeding up query processing in Big data framework is defined as a multi-objective optimization problem in this Chapter. Implementation of a modified version of Multi-objective Differential Evolution (MODE) algorithm with binary encoded data and implementation of Non-dominated Sorting Genetic Algorithm -II have been discussed in this chapter with their performance analysis in handling this problem.

- **Chapter 5**: *Multi-objective Simulated Annealing Algorithm in Big data View Selection for Materializing* - This chapter presents how multi-objective Simulated Annealing algorithm based techniques may be applied in selecting sub-query results or views in MapReduce based query processing framework. A comparative performance analysis of this technique with respect to common EA based techniques in view selection problem in this paradigm has also been presented in this chapter.

- **Chapter 6**: *Conclusion and Future Direction* - This Chapter concludes the dissertation by summarizing the overall contribution and identifies some future directions of research in this area.