DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2017

BigDataCube: Distributed Multidimensional Data Cube Over Apache Spark

An OLAP framework that brings Multidimensional Data Analysis to modern Distributed Storage Systems

WEHERAGE, PRADEEP PEIRIS

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

BigDataCube: Distributed Multidimensional Data Cube Over Apache Spark

An OLAP framework that brings Multidimensional Data Analysis to modern Distributed Storage Systems

Weherage, Pradeep Peiris

Master of Science Thesis

Software Engineering of Distributed Systems
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

1 August 2017

Examiner: Seif Haridi
Supervisor: Amir Payberah

TRITA-ICT-EX-2017:125

© Weherage, Pradeep Peiris, 1 August 2017

Abstract

Multidimensional Data Analysis is an important subdivision of Data Analytic paradigm. Data Cube provides the base abstraction for Multidimensional Data Analysis and helps in discovering useful insights of a dataset. On-Line Analytical Processing (OLAP) enhanced it to the next level supporting online responses to analytical queries with the underlying technique that precomputes (materializes) the data cubes. Data Cube Materialization is significant for OLAP, but it is an expensive task in term of data processing and storage.

Most of the early decision support system benefits the value of multidimensional data analysis with a standard data architecture that extract, transform and load data from multiple data sources into a centralized database called Data Warehouse, on which OLAP engines provides the data cube abstraction. But this architecture and traditional OLAP engines do not hold with modern intensive datasets. Today, we have distributed data storage systems that keep data on a cluster of computer nodes, in which distributed data processing engines like MapReduce, Spark, Storm, etc. provide more ad-hoc style data analytical capabilities. Yet, there is no proper distributed system approach available for multidimensional data analysis, nor any distributed OLAP engine is available that follows distributed data cube materialization.

It is essential to have a proper Distributed Data Cube Materialization mechanism to support multidimensional data analysis over the present distributed storage systems. Various research work available today which considered MapReduce for data cube materialization. Also, Apache Spark recently enabled CUBE operator as part of their DataFrame API. The thesis raises the problem statement, the best-distributed system approach for Data Cube Materialization, MapReduce or Spark? and contributes with experiments that compare the two distributed systems in materializing data cubes over the number of records, dimensions and cluster size. The results confirm Spark is more scalable and efficient in data cube materialization than MapReduce. The thesis further contributed with a novel framework, *BigDataCube*, which uses Spark DataFrames underneath for materializing data cubes and fulfills the need of multidimensional data analysis for modern distributed storage systems.

Sammanfattning

Multidimensional Data Analysis är en viktig del av Data Analytic paradigm. Data Cube tillhandahåller den grundläggade abstraktionen för Multidimensional Data Analysis och hjälper till att hitta användningsbara observationer av ett dataset. On-Line Analytical Processing (OLAP) lyfter det till nästa nivå och stödjer resultat från analytiska frågor i realtid med en underliggande teknik som materliserar Data Cubes. Data Cube Materialization är signifikant för OLAP, men är en kostsam uppgift vad gäller processa och lagra datat.

De flesta av tidiga beslutssystem uppfyller Multidimensional Data Analysis med en standarddataarkitektur som extraherar, transformerar och läser data från flera datakällor in I en central databas, s.k. Data Warehouse, som exekveras av OLAP och tillhandahåller en Data Cube-abstraktion. Men denna arkitektur och tradionella OLAP-motorer klarar inte att hantera moderna högbelastade datasets. Idag har vi system med distribuerad datalagring, som har data på ett kluster av datornoder, med distribuerade dataprocesser, så som MapReduce, Spark, Storm etc. Dessa tillåter en mer ad-hoc dataanalysfunktionalitet. Än så länge så finns det ingen korrekt angreppsätt tillgänlig för Multidimensional Data Analysis eller någon distribuerad OLAP-motor som följer Distributed Data Cube Materialization.

Det är viktigt att ha en korrekt Distributed Data Cube Materialization-mekanism för att stödja Multidimensional Data Analysis för dagens distribuerade lagringssystem. Det finns många forskningarar idag som tittar på MapReduce för Data Cube Materialization. Nyligen har även Apache Spark tillgänglitgjort CUBE-operationer som en del av deras DataFrame API. Detta examensarbete tar upp frågeställningen, vilket som är det bästa angrepssättet för distribuerade system för Data Cube Materialization, MapReduce eller Spark. Arbetet bidrar dessutom med experiment som jämför de två distribuerade systemen i materialiserande datakubar över antalet poster, dimensioner och klusterstorlek. Examensarbetet bidrar även med ett mindre ramverk BigDataCube, som använder Spark DataFrames i bakgrunden för Data Cube Materialization och uppfyller behovet av Multidimensional Data Analysis av distribuerade lagringssystem.

Acknowledgements

I would like to express my sincere gratitude to Dr. Amir H. Payberah, my supervisor, for the continuous support and guidance. His knowledge in the area of data intensive computing and technical background in frameworks like Apache Spark led me to narrow down the problem area and identify the possible contributions of the thesis. Also, I am thankful to Professor Seif Haridi for his valuable feedback to improve the thesis work.

I am deeply grateful to Magnus Danielsson giving me the work opportunity and allowing me to carry out the thesis work at Digital River World Payment. The business domain and its technology stack in Digital River World Payment helped me to recognize a good project proposal for the thesis. I would like to thank Ravinath Senevirathne for taking the time to proofread my thesis and for giving me valuable feedback. Also Thérèse Gennow, for her help in writing Swedish abstract for the thesis.

Contents

Chapter 1

Introduction

Data Analysis is a broad subject, which involves the process of collecting, cleaning and transforming data into *Analytical Models* that assist discovering useful insight and new knowledge from the data. The analytical models include many advanced techniques such as Statistics, Data Mining, and Machine Learning.

In contrast to advanced analytical models, more straightforward but yet powerful data analytical technique include the Relational model and its Structured Query Language (SQL). Many useful data analytical questions can be composed of SQL and its data aggregation functions. Most of the early enterprise applications widely used this approach for their Decision Support Systems. But the relational model is solely designed for highly transactional systems with efficient data storage. The relational systems by its design are inadequate for data analysis purposes. *Multidimensional Data Modeling* emerged during 1990 to fulfill this need.

The Multidimensional Data Modeling does not originate from database technologies. As a manual data analysis tool based on multidimensional matrix algebra, it dates back to late 19th century [39, p.2]. The basic notion of the multidimensional modeling is to define a data relationship model among all possible data attributes in a data collection. For example, consider the dataset about the number of car sales given in Table 1.1 *. The possible data relationships appear in the dataset are Number of Sales per Model, Number of Sales per Year, Number of Sales per Model and Year, etc. Multidimensional data modeling classifies data attributes into two main categories; *Dimensions* and *Measures*. The measures represent quantitative attributes countable with the combination of dimensional attributes. In the sample dataset, Number of Sales represents a measure field, whereas Year, Model and Color stand for dimensional fields.

The multidimensional model achieves its data relationship modeling with the

* It is the sample dataset that Jim Gray et al. used for describing CUBE operator in their paper [30].

CAR SALES			
Model	Year	Color	No. of Sales
Chevy	1990	red	5
Chevy	1990	white	87
Chevy	1990	blue	52
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	blue	49
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	blue	71
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	blue	63
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	blue	55
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	blue	39

Table 1.1: Factual data collection with three dimensions and one measure

arrangement of data in a multidimensional data structure called hypercube or commonly known *Data Cube*. Figure 1.1 illustrates how dataset given in Table 1.1 is organized in a 3-dimensional data cube. Each cell in the data cube holds the measures of the dataset, which correlates with its dimensional values. Organizing data in a multidimensional data cube provides an easily understood conceptual view for the dataset. Also, it provides fast access to any data points; measures by its dimensional values.

The base data cube in figure 1.1 can be further transformed into many other useful data cubes that fulfill various data analytical questions. For example, one might be interested in number for sales for any Models but per Year and Color. This is simply achieved with shrinking the data cube toward the Model dimension. Figure 1.2 illustrates the resulted cube after the data cube transformation. Another might be interested in the number of sales per Year and Color but for a selected Model, *ford*. This can be achieved with slicing out the data cube over the selected dimensional value, Figure 1.3 shows corresponding cube transformation. In this way, a data cube can be transformed into many other data cubes that provide

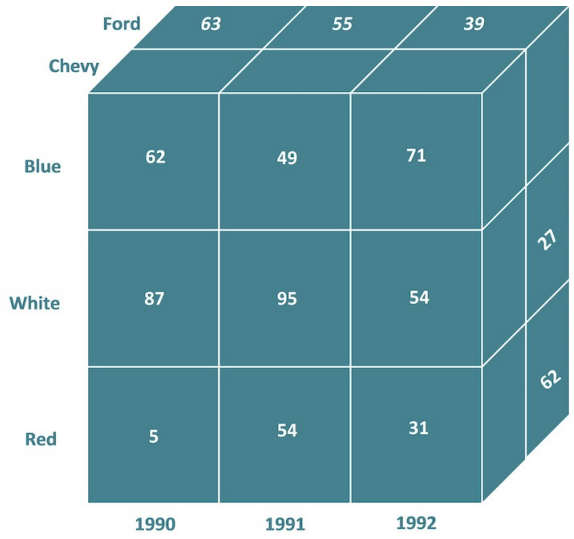


Figure 1.1: Data Cube for the dataset given in table 1.1

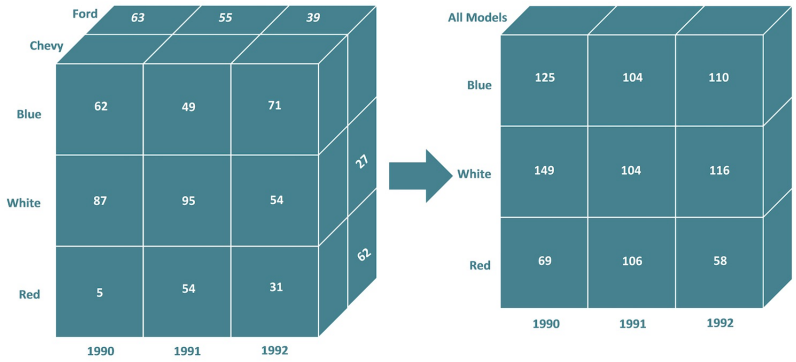


Figure 1.2: Data Cube transformation: Roll-up the cube toward Model dimension

useful insight of a dataset. The data cube transformations determine a set of *Cube Operations* or *Cube Algebra* that operates on data cubes [20, 44].

1.1 Motivation

Data cube provides the fundamental building block for multidimensional data analysis. It allows users to build valuable insights of the data with various data cube operations. The cube operations include *slice* and *dice* that acts on a subset of data cube, *drill down/up* that navigates through dimensional hierarchies, aggregations that summarize along dimensions, and *pivot* that rotates a cube to show its different views.

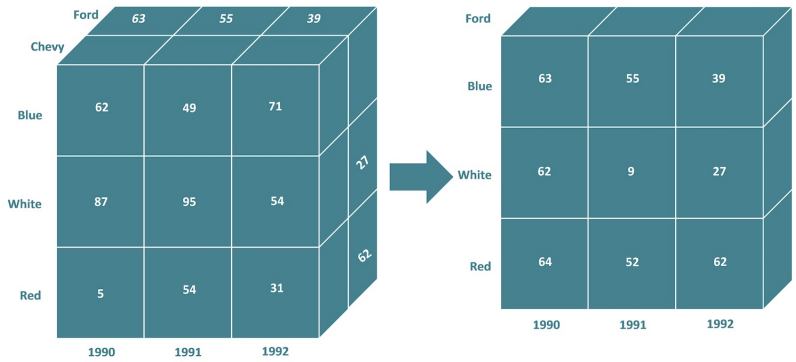


Figure 1.3: Data Cube transformation: Slice the cube over a selected dimension

E. F. Codd brought multidimensional data analysis to the next level with the idea of *On-line Analytical Processing* (OLAP) [24]. It considers providing online responses to analytical queries over the multidimensional model. The primary technique for achieving fast query response is to precompute or *materialize* all possible data cubes. Hence the analytical queries can be responded instantly without performing any data cube operations at runtime. But data cubes are intrinsically large. In the real world, they contain many dimensions, each of with high cardinality levels. Therefore, *Data Cube Materialization* involve the challenge of precomputing all possible massively large data cubes. From the sample dataset in Table 1.1, eight data cubes are producible with the combination of dimensions; Model-Year-Color, Model-Year, Model-Color, Year-Color, Model, Year, Color and None. Figure 1.4 illustrates all these data cubes. The number of computable data cubes gets doubled if we introduce a single dimension to the dataset. That is, the number of possible data cubes grows exponentially with the number of dimensions. A dataset with n dimensions engenders the challenge of constructing 2^n number of data cubes. Therefore, data cube materialization is expensive in both computation and storage, which brought a new research problem to the community, to find efficient techniques for data cube materialization.

Most of the early research work in data cubes and OLAP methodologies considered full cube materialization [30, 2, 80] with efficient grouping algorithms. But later research work considered the full cube materialization is inefficient in both computation and storage for the high-dimensional cubes. They proposed several other approaches with the following main directions;

Partial Cubes: Only subset or part of the cubes are materialized. Iceberg Cube [26, 13, 33, 60, 77], BPUS [34] and PBS [62, 10].

Compressed Cubes: Algorithms and models that reduce the size of the cubes. QuantiCubes [27], Quotient Cube [47, 46], Condensed Cube [74], Closed

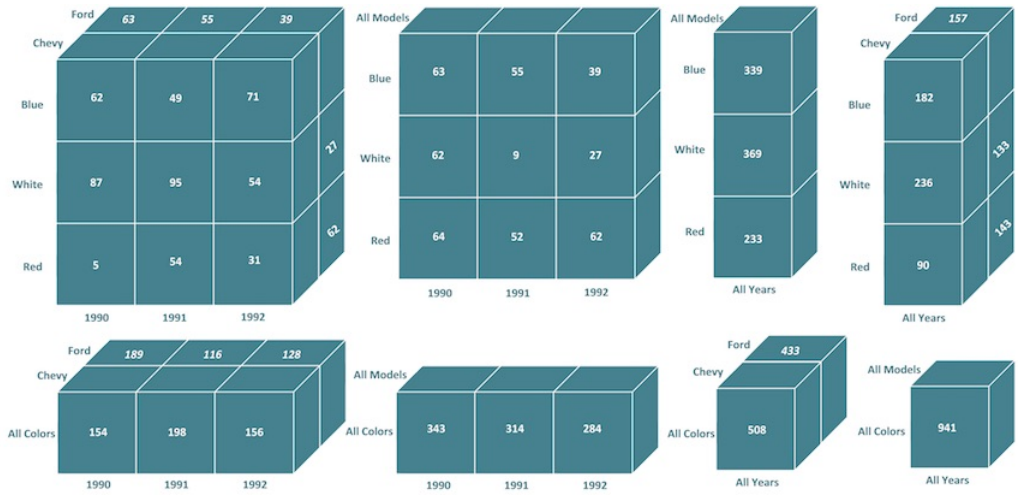


Figure 1.4: All possible Data Cubes for the dataset in table 1.1

Cube [76], Dwarf Cube [64] and MDAG Cube [16].

Approximated Cubes: Provide approximate answers to OLAP queries. Quasi Cube [23, 11], Wavelets [72, 73] and Loglinear Cube [12].

All these research directions considered efficient techniques for data cube materialization but with the resource constrained in a capacity of single computation node. At the same time, various parallel processing techniques were proposed [29, 41, 22, 18] that improve the data cube materialization with maximum resources usage of single computation unit. But none of these techniques scale with modern intensive data flows. The applications today consider Big data that spans over a hundred of computation nodes. The data analytical capabilities are expected over this continually growing exhaustive data loads. Therefore, the previous works on data cube materialization should be adapted to Distributed System Models to sustain the scalability.

Fortunately, *MapReduce* [37] is well established today as a distributed computation model for various data intensive computation problems. Several researchers have considered data cube materialization over MapReduce [1, 75, 48, 52]. It resolves two fundamental problems in data cube materialization; Storage, over the *Distributed File System*, and Computation, taking it close to the data with MapReduce. But some studies show MapReduce is inefficient for applications that reuse intermediate results across multiple computations (e.g., Iterative machine learning, graph processing, and interactive data mining). Resilient Distributed Datasets (RDDs) [79] was introduced to overcome this major issue. RDD is a distributed memory abstraction that allows in-memory computation over a large cluster of nodes. It resolves the problem in MapReduce

by keeping data in memory for multiple computations. Apache Spark [6] employs RDDs providing sub-frameworks such as Streaming, MLib, GraphX and most importantly Spark SQL [9].

Spark SQL is a new module in Apache Spark that leverage the benefits relational processing over the Spark RDD. It offers a declarative DataFrame API with a highly extensible optimizer called Catalyst. Apache Spark introduces CUBE operator as a part of DataFrame API in version 1.4.0. That is, it creates all possible data cubes for a given DataFrame. It simplifies the problem of data cube materialization over the distributed storage systems.

1.2 Problem Description

Data cube provides a conceptual model for multidimensional data analysis. The classical cube operations; *Slice*, *Dice*, *Drill Down/Up*, and *Aggregation* are reasonable enough to express many useful analytical queries. OLAP brought multidimensional data analysis to the next level supporting online responses on analytical queries. But multidimensional data analysis or OLAP is rarely discussed in modern Big Data Analytic paradigm.

1.2.1 Problem

There is no standard OLAP engine available today that support multidimensional data analysis over distributed data collections. Data cube materialization is critical for such a distributed OLAP engine that enables multidimensional data analysis in Big Data analytic paradigm. Data cubes are large and computationally expensive. But with the modern intensive data loads, they can be even immensely large and expensive. It requires an efficient distributed computing approach for materializing data cubes. Two leading directions of distributed data cube materialization are MapReduce and CUBE operator of Spark's DataFrame API. There is no research study available that compares MapReduce and Spark DataFrame CUBE operator in data cube materialization. *It is important to select the most efficient distributed data cube materialization approach for a standard OLAP engine that enables multidimensional data analysis over distributed data storage systems.*

1.2.2 Hypothesis

MapReduce is a widely used distributed computation model for various data intensive computation problems. RDD in Apache Spark emerged to address one of the main issues in MapReduce, its inefficiency in data reuse for iterative data

processing algorithm. Data cube materialization formulates the lattice model [34], a hierarchy of cubes (Lattice diagram), which states the data cubes in the lower hierarchy can be constructed from the cubes in the previous level in the hierarchy. Therefore, Apache Spark's RDD would provide the best-distributed computing model for data cube materialization. It could be even improved with DataFrame API in Spark that uses additional optimization techniques. *We hypothesize CUBE operator of DataFrame API in Apache Spark provides the best-distributed computing abstraction for Data Cube Materialization and it outperforms MapReduce.*

1.3 Contribution

There are three main contributions of the thesis work. Firstly, the experiments and its results that validate the hypothesis of the thesis, that CUBE operator of DataFrame API in Apache Spark provides the most efficient distributed computing approach for Data Cube Materialization.

Secondly, the design and implementation of the framework *BigDataCube*, a distributed On-line Analytical Processing engine with Apache Spark. It enables multidimensional data analysis over distributed storage systems such as HDFS, Cassandra, Hbase, etc., almost any distributed storage systems that Apache Spark supports. Also, the framework includes with a Pivotal View, a front-end data analysis tool that presents multidimensional data cubes in pivot tables, which enables users (Data Analysist, Decision Makers, Managers, etc.) to aid any useful insight of the data.

Thirdly, the utilization of *BigDataCube* framework on Payment Gateway Platform in *Digital River*[*]. The *BigDataCube* framework is applicable over any business domain. Here we consider the Payment Gateway Platform as a use-case for applying the framework and enabling multidimensional data analysis over Apache Cassandra. It brings a new service component to the platform in Digital River and enables Multidimensional Data Analysis.

1.4 Ethics and Sustainability

The thesis peruses Multidimensional Data Analysis in modern distributed storage system. As it involves data in large-scale and discloses new insights from the data, the awareness of ethical factors of the new insights is very important. One

[*] Digital River, the organization at which the Master's thesis was carried out, is a service provider for global online payments. The standardized payment gateway allows integration of many merchants into the Payment Platform and routes their transaction over various Payment Networks.

can simply reveal unintentional information that harms to third-party from a multidimensional data model. For example, the thesis aims the application of its new framework, BigDataCube in the online payment gateway at Digital River, which receives payment transactions from merchants in diverse business domains. Internally, it may be an important factor for Digital River to rank merchants from their transactions count and value, but it could be a huge business impact in case such information was disclosed to outside. Therefore, as a data analytical tool, BigDataCube should be access granted to relevant parties in the organization.

Sustainability determines economics, social, political and technological boundaries that make things better for more people. One of the contributions of the thesis is to find an efficient distributed computing approach for Data Cube Materialization. Distributed computing systems with efficient resource usage and faster processing power always decrease the operational cost of data centers, hence benefit the environment and society. In addition, the thesis contributes with a framework, BigDataCube, which is reusable for any business domains with minimum configuration changes. That is, the thesis not only contributes a multidimensional data analysis tool with efficient distributed computing approach but also a reusable tool for many organizations and enterprises.

1.5 Outline

The rest of the thesis report is structured as follows:

Chapter 2. Background-Multidimensional Data Analysis provides a detailed background of Multidimensional Data Analysis and Data Cube abstraction. Then it states how OLAP emerged in the area of multidimensional data analysis and convenience of Data Cube concept in OLAP.

Chapter 3. Data Cube Materialization and Related Work covers the importance of Data Cube Materialization and its challenges in multidimensional data analysis and OLAP. It further explains and categorizes early research works in Data Cube Materialization.

Chapter 4. Distributed Data Cube Materialization emphasizes early works in Data Cube Materialization is inapplicable with modern large-scale data collections and their Distributed Storage technologies. It presents the theoretical and technical background of Data Intensive Computing and related work in Distributed Data Cube Materialization.

Chapter 5. Apache Spark vs Hadoop in Data Cube Materialization provides the answer to the one of problem statement in the thesis, the best-distributed

computing approach for Data Cube Materialization, Spark DataFrame API or MapReduce? It presents the research methodology used to conclude the problem statement and its outcome.

Chapter 6. BigDataCube: A framework for Multidimensional Data Analysis with Apache Spark explains the main contribution of the thesis. BigDataCube is a framework that uses CUBE operator of DataFrame API in Apache Spark for Data Cube Materialization. It takes a design approach to support many distributed data storage systems and provides configuration options to select Dimensional and Measure fields for the target data cubes. It further explains how embedded Pivot View in the framework visualizes the selected data cube.

Chapter 7. Case Study: Big Data Cube overApache Cassandra is another contribution of the thesis, the usage of the framework in Digital River's Payment Gateway Platform. It enables Multidimensional Data Analysis for all transactional payment data reside in Cassandra and help Business Analysts and Decision Makers in Digital River to gain insight of the transaction data.

Chapter 8. Conclusion and Future Works summarizes the whole work in the thesis and its key results. It explains how BigDataCube framework can be further improved and potential future work in the area of Data Cube Materialization, OLAP and Multidimensional Data Analysis.

Chapter 2

Background - Multidimensional Data Analysis

Multidimensional data analysis is one of widely used data analysis approach in many enterprise application domains. The hypercube or more commonly known Data Cube provides a Multidimensional Analytical Model for data analysis. The data cube organizes factual information as Dimensions and Measures. The Measures represent numerical properties of factual information, which are explained and selected via associated dimensions. The high-level Cube Operations; Roll-up, Drill-down, Drill-across, Slice, and Dice allow querying multidimensional data for various data analysis questions. The On-Line Analytical Processing (OLAP) led data analysis into the next level providing fast query responses for complex data analysis questions.

2.1 Multidimensional Data Analysis

Data Analysis involves the process of collecting, cleaning and transforming data into analytical models that assist understanding useful insight of the data or extracting new knowledge out of the data. A typical data analysis project is built in multiple phases, which may vary from project to project. Thomas A. Runkler in his book [58], summarizes a typical data analysis process into four main stages;

Preparation The preparation phase mainly considers data collection, usually from multiple data sources that may represent diverse data platforms and systems. The data from heterogeneous systems are vast and complex. Therefore, it is important to select only the relevant data for the problem.

Pre-possessing Data from multiple sources may follow different formats and standardizations. The quality of the data cannot be guaranteed. In

prepossessing, the data is cleaned, filtered, corrected and transformed into the target format and standardization.

Analysis In this phase, the rectified data are applied to Analytical Data Models that gain useful insight from the data.

Post-processing The final phase usually considers the evaluation of the result, interpretation, and documentation.

It is the analysis phase essential in data analysis projects. It comprises algorithms that reform data into Analytical Models, from which useful insight of the data can be achieved. The analytical models include more advanced techniques such as Statistical and Data mining models. The statistical models mainly focus on discovering underlying patterns and trends of the data. There are two main approaches to statistical modeling. One method assumes data are generated by stochastic models (linear regression, logistic regression, etc.), while other approach uses algorithmic models that assume data mechanism is complex and unknown (decision trees, neural nets) [15]. Data mining is an interdisciplinary subject, which also combines statistical models with other techniques in discovering new knowledge from data. The popular data mining models include Clustering, Classification, Correlation, etc.

Apart from advanced data analytical models, more straightforward and but yet powerful data analysis approaches include simple query and aggregation functions. The well-known example is Structured Query Language (SQL) and its various SQL functions. Many data analytical questions can be issued with SQL queries over a relational database model. There exist many data analytical and visualized tools [67, 54] that provide a simplified API, which transforms analytical requests into SQL queries to extract data from the underline relational database model. SQL and Relational data model provide a powerful tool for analyzing a dataset, but the relational model was solely designed for fast transactional processing with efficient data storage. It was E.F. Codd introduced the relational data model [19] in 1970, which provides the foundation for many Database Management Systems exist today. As Gavin Powell explains in his book [56, p.9], the relational database model began as a way of getting groups of data from a larger dataset. It removes any data duplications and increases the granularity of data elements. As further explained in the book [56, p.173], the relational model is efficient for operational, or On-line Transaction Processing (OLTP) applications with fast data processing requirements, which involve frequent and concurrent data access over a small set of data by many users.

Even though the relational model was originally designed for highly transactional systems, it is often used for various data analysis scheme. The idea of using relational model for data analysis purposes has been further extended in the

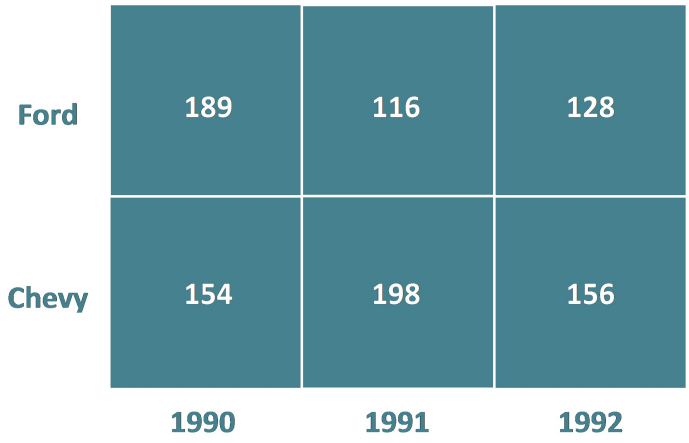


Figure 2.1: Number of Sales in two dimensions; Year and Model

application area of *Decision Support Systems*. Many early enterprises promoted the approach of storing data redundantly in two separate systems; one for transactional processing systems and the other for decision support systems. As the author of the relational data model, E.F. Codd considers these approaches are imprecise and fuzzy that ignores the fundamental requirements of the analytical data models in his late paper [24]. Thus, it requires another data model for data analysis. The multidimensional data model emerged during 1990 to fulfill this need. It considers supporting complex data analysis functions over a large dataset.

2.2 Data Cube

The multidimensional data models do not origin from database technologies. It originates from multidimensional matrix algebra and has been used in manual data analysis since late 19th century [39, p.2]. Multidimensional data are more natural to represent in a multidimensional array.

There are typically a number of different dimensions from which a given pool of data can be analyzed. This plural perspective, or Multidimensional Conceptual View appears to be the way most business persons naturally view their enterprise [24].

— E.F. Codd

The Multidimensional Conceptual View, E. F. Codd mentioned here became the core data model for multidimensional data analysis, and was commonly known as Multidimensional Data Cubes. The arrangement of data in a cube naturally

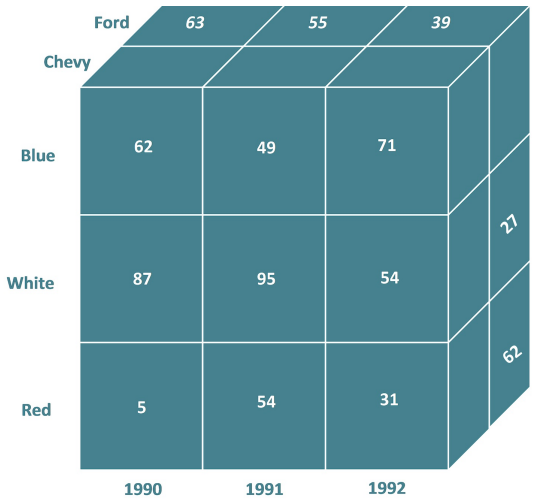


Figure 2.2: Number of Sales in tree dimensions; Year, Model and Color

fits how human used to analyze a dataset manually. Let us consider the sample dataset given in table1.1. The figure 2.1 shows how the data collection is arranged in a two-dimensional cube. The dimensions selected for the cube are *Year* and *Model*. The cells contain the measures; Number of Sales per Year and Model. Organizing data in this way enables a way to understand how data is distributed over the dimensions and provides fast access to any data points (cells). And it leads to a simplified model for any complex data analytical questions. For example, Number of sales related to any Model can be easily filtered out with selected dimensional value for Model. The total sales for a given Year can be easily calculated aggregating all date indexes of the Year.

Now consider, we want to see the breakdown of the sales by Color. That is an introduction of a new dimension. Figure 2.2 shows the tree dimensional view of the sales. As it depicts in the diagram, each cell now represents the number of sales per Year, Model and Color. In a similar way, the measure can be broken down into any possible dimensions.

Dimensions: The dimensions provide a way for selecting measures. In other words, each data point in a data cube describes a set of dimensional values. Also, there is another important purpose and use of dimensions. That is grouping measures into multiple levels. Take for an example the Year dimension in the example. The Year can be further broken down into its Quarter, Month, Week and Day. In dimension, these attributes are called Levels. Figure 2.3 shows the different levels of the time dimensions. In a similar way, any dimension may have one or more dimensional levels, and it helps further aggregation of date cubes.

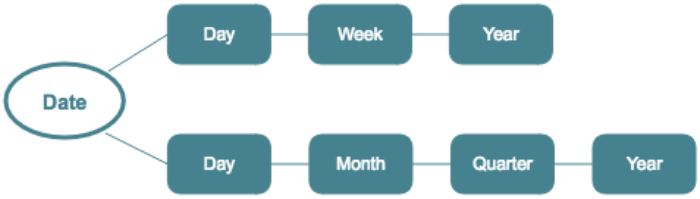


Figure 2.3: Levels in Date dimension

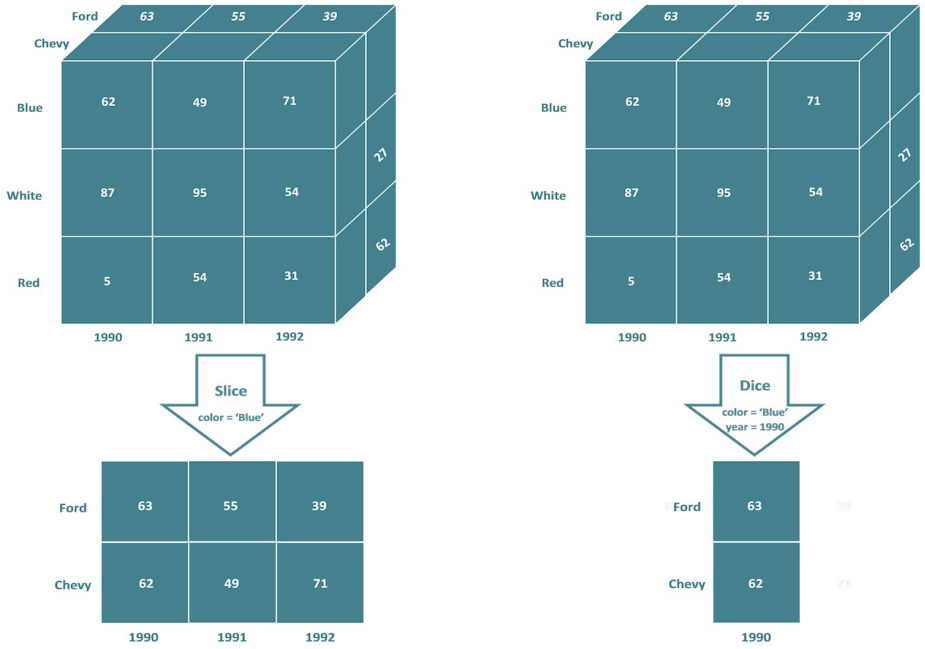
Measures: A combination of dimensions determines a data point in the cube, which has one or more measures. From the data cube in figure 2.2, the data point at (1990, Chevy, Red) has the value 5, which represents the measure; the number of sales. The measures can be in two categories. Firstly, the numerical data aggregation functions such as Summation, Average, Maximum, Minimum, etc. that determine measure value for the given combination of dimensions. Secondly, any mathematical formulas that generate new measures with a combination of other measures.

2.3 Data Cube Operations

Organizing data in a multidimensional data cube provide the base mean for data analysis. As SQL provides a way for querying structured data in a relational model, the Cube Operation provides a way for querying data in a cube. A set of Cube Operations explains, which allow different views of cubes and interactive analysis and querying. These are Roll-up, Drill-down, Drill-out, Drill-across, Slicing and Dicing operations that can be applied on a data cube [39, pp. 18-23]. These cube operations in general considered as closed operations, as the application of these operations on cubes result in another new cube.

Slice and Dice: The slice operation applies on a cube with a selected single dimensional value. It results in a subcube with one less dimension from the original cube. Figure 2.4a shows the result of applying slice operation on the Color dimension where the color value is 'blue'. In contrast to slice operation, the dice operation used with two or more dimensions. Diagram2.4b shows the further filtering on sales cube with dice operation with dimensional value Color and Year.

Roll-Up and Drill-Down: There is two variant of Roll-Up operation; Dimensional Level roll-up and Dimensional Reduction roll-up. As it depicts in Figure



(a) Slice operator (b) Dice operator

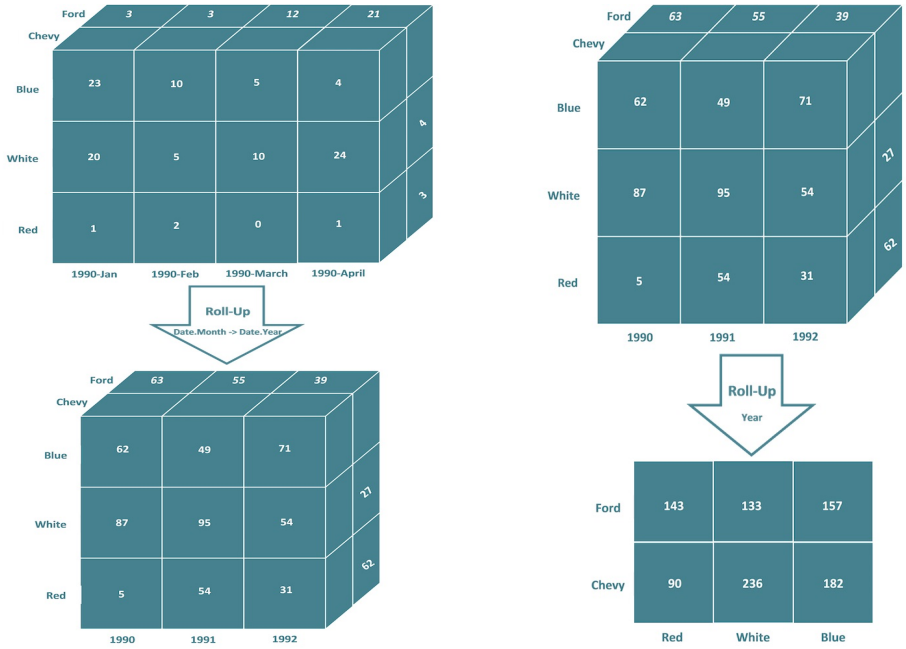
Figure 2.4: Slice and Dice operators

2.5a, in Dimensional Level Roll-up, the measures are further aggregated toward dimensional level but end up with a cube with the same number of dimensions. In Dimensional Reduction Roll-up, the measures are aggregated with fewer dimensions as it is in Figure 2.5b. The Drill Down operator is the opposite of Roll-Up operator. That is, it navigates toward down the dimension hierarchy or introduce with a new dimension.

Drill Across The Drill across operation combines two cubes over shared dimensions and produce a new cube.

2.4 Online Analytical Processing (OLAP)

Over the time with the growth of data, many organizations recognized the business value of the operational data for their decisions of activities. The advent of decision support systems required data from these multiple data sources to be captured into a centralized system to support more effective decision support queries. Thus the concept of Data warehouse emerged. Unlike operation data sources, the data warehouses contain an enormous amount of data, usually



(a) Roll-Up: Dimensional Level

(b) Roll-Up: Dimensional reduction

Figure 2.5: Roll-Up operator

historical data that span several years, and decision support queries commonly considered as expensive and time-consuming. But the demand for decision support system increased, and it was crucial to have decision support queries with recent response time. The multidimensional data model; Data Cube was recognized as the best data model for Decision support systems, and E.F. Codd brought it to the next level with the newly coined term Online Analytical Processing (OLAP) [24], which mainly considers providing timely response to any complex decision support queries.

Figure 2.6 illustrates typical system architecure of Decision Support Systems. As it depicts data from multiple Data Sources are extracted, transformed and loaded (ETL) into a target Data Warehouse System. The data sources usually include relational database management systems that follow the normalized relational model to support online transactional systems. The data sources even can be any file management system. The Data Warehouse is a central data repository on which any analytical queries are issued. The Data Warehouses are usually implemented in Relational or Multidimensional Database Management Systems where the physical model is optimized for data reading. The ETL is the data integration process in between Data Sources and Data Warehouse. The standard approach in many ETL implementations is to extract source data to a

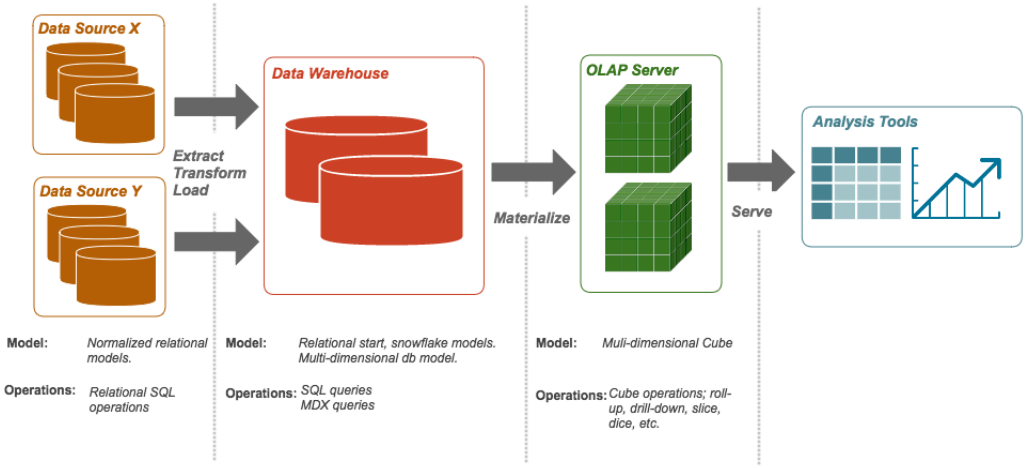


Figure 2.6: Typical Multidimensional Data Analytics Architecure

staging area of Data Warehouse Management System. The data separation in this way reduces any impact on the source transactional application from the ETL processes. The data collected in staging areas come from different data sources that may follow different data formats, type, standardization, etc. It is part of the ETL process to transform all the data into a standardized scheme that supports the target Data Warehouse System. Finally, the ETL process uploads all transformed data into the target Data Warehouse System. As it depicts in the figure, OLAP is a new architectural component stands in between Data Warehouse and Analytical Tools, that supports merely online responses to decision support queries. Various OLAP products [51, 42, 53] appeared during the time that fit into this architecture.

A swiftly response time for OLAP queries is achieved with pre-computation or materialization of all possible data cubes. The Data Cubes are intrinsically large, and materialization of all possible data cube is a nontrivial task in term of storage and computation. This became one of focusing research problem in OLAP. Many research and industrial communities have taken various approaches for efficient cube materialization since the introduction of OLAP.

Chapter 3

Data Cube Materialization and Related Work

Data Cube provides an adequate model for Multidimensional Data Analysis, which involves decision support queries that require heavy use of data aggregations. The OLAP brings Multidimensional Data Analysis to the next level supporting timely response on decision support queries. Providing a swift response on OLAP queries require precomputation or materialization of all possible data cubes. But, Data Cubes are intrinsically large and grow exponentially with the number of dimensions and its cardinality level. Also, computationally expensive against the number of dimensions. These factors bring a new research problem to the community; Data Cube Materialization with efficient storage and computation.

Most of the early research proposals considered full cube materialization with efficient grouping algorithms [30, 2, 80]. With growing dataset and high dimensional data domains, the full cube materialization was determined inefficient and impractical. Therefore, several other approaches were proposed during the time. In one research direction, data cube compression was considered with models such as QuantiCubes [27], Quotient Cube [47, 46], Condensed Cube [74], Closed Cube [76], Dwarf Cube [64] and MDAG Cube [16]. Another research direction was to select part of the cube for the materialization; Iceberg Cube [26, 13, 33, 60, 77], BPUS [34] and PBS [62, 10]. With a naive assumption that data analysis does not require an accurate result, approximate data cube computation was engaged in types of research; Quasi Cube [23, 11], Wavelets [72, 73] and Loglinear Cube [12]. Also, some researchers improved previously proposed algorithms with parallel computation models [29, 41, 22, 18].

In general, early researchers on data cubes can be categorized into 4 main topics; Full data cube computation with efficient algorithms, Data cube compression, Partial cubes and Approximate cubes. The *Data Cube* operator [30] and *Lattice framework* [34] provided the base semantic in all of these research directions.

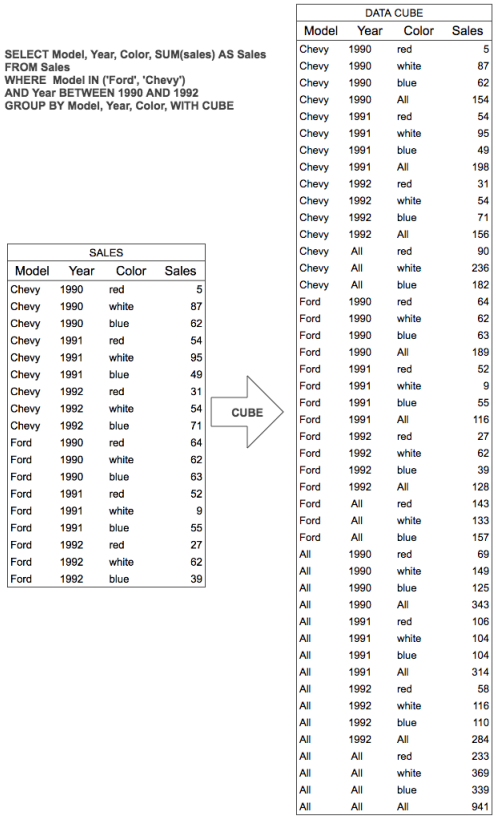


Figure 3.1: Example of DATA CUBE Operator

3.1 Full Cube Materialization

Early research work on data cube materialization considered the calculation of full cube. That is the materialization of all possible data cubes with efficient data grouping methods. It is the CUBE relational operator [30] first initiate the idea of computing of all possible data cubes.

3.1.1 Data CUBE Operator

Jim Gray et al. proposed a relational aggregation operator called *CUBE*, which generalized all possible *GROUP BY* relational operator over n-dimensional attributes [30]. Figure 3.1 shows an example taken from the original paper [30] how CUBE operator is applied on a relational dataset and its result. As it can be seen in the figure, the CUBE operator calculates GROUP BY over all possible dimensional combinations. Each dimensional combinations produce a data cube, which is known as *cuboid* [32, pp. 156-158] (From this onward, we use the term

'cuboid' to mean a single Data Cube). OLAP uses different cuboids for various analytical queries. Therefore having all possible cuboids pre-computed leads for timely query response time.

3.1.2 PipeSort and PipeHash

S. Agarwal et al. proposed two algorithms [2] to achieve CUBE operator. The algorithms; *PipeSort* and *PipeHash* view the structure of CUBE computation as a hierarchy of group-by operations, which improved a factor of two to eight in computing CUBE with separate group-bys.

3.1.3 Multidimensional Arrays

In contrast to ROLAP approaches based on CUBE operator, Y. Zhao, P. M. Deshpande, and J. F. Naughton proposed data cube computation approach for Multidimensional OLAP systems, which store data in physical sparse arrays. The proposed *MultiWay* array methods resulted in better performances than previous ROLAP methods in term of both CPU and storage.

The multidimensional array based techniques (MOLAP systems) found efficient in computing cuboids with the fast random accessing capability in arrays. But, it was a limitation that the size of the arrays at each dimension required to be fixed and need the reallocation of all elements in the arrays in case introduction of new dimensional value. T. Tsuji, A. Hara, and K. Higuchi proposed an incremental cube computation [69] for MOLAP by employing extendible arrays, which can be extended dynamically in any direction without any data reallocation. D. Jin, T. Tsuji, and K. Higuchi took it to the next level using a single extendible array to manage full cube incrementally [40].

3.2 Partial Cube Materialization

Many researches considered full data cube materialization does not scale in term of computation and storage space. The number of cuboids growth exponentially against the number of dimensions in a model. For example, consider a 3 dimensional model with dimensions d_1, d_2, and d_3. The possible combination of group-by or cuboids are $d_1d_2d_3$, d_1d_2, d_2d_3, d_1d_3, d_1, d_2, d_3 and *All*. In this way, the 3-dimensional model produces 8 possible dimensional combinations. That is n-dimensional model results computation of 2^n number of cuboids. But in practice dimensions are expressed in a hierarchy of multiple levels. This even makes the calculation of more cuboids. The following formula from paper [30] expressed the total number of cuboids required for n-dimension model, where L_i represents

the number of levels associated with the dimension i. Also, it is expensive in term of storage. The number of entries or tuples created from each cuboid depends on the cardinality of the cuboid.

$$\textit{Total number of cuboids} = \sum_{i=1}^{n} (L_i + 1) \tag{3.1}$$

This led to another research direction that consider the materialization of the subset of cuboids that is relevant for OLAP queries. The lattice framework [34] made the foundation for this research direction.

3.2.1 The Lattice of Cuboids

The calculation of cuboids is an expensive process, and it is impractical to materialize all the possible cuboids. This led research direction to the partial materialization of the cubes. V. Harinarayan, A. Rajaraman, and J.D. Ullman proposed the lattice framework, which states that some cuboid can be computed from the result of another cuboid [34]. They generalized the problem in a lattice diagram which shows dependencies of cuboids. Figure 3.2 shows the lattice diagram for 3 dimensional model with dimensions d_1, d_2, and d_3. As it depicts in the figure, cuboids d_1d_2, d_2d_3 and d_1d_3 can be computed from the cuboid $d_1d_2d_3$. The cuboid $d_1d_2d_3$ is called *based cuboid* as the all other cuboids can be calculated from it. The cuboid d_1 can be computed from both d_1d_2 and d_1d_3. Finally, the *none* cuboid can be computed from any of d_1, d_2 or d_3, which is called *apex cuboid* [32, pp. 157-158].

The lattice framework was attractive among many other researches, and it became the base approach for materializing cuboids. The authors of the paper [34] also contributed with a greedy algorithm, which selects best cuboids for materialization from the lattice framework.

3.2.2 BPUS, PBS Algorithms

The lattice framework [34] expresses dependencies between cuboids. V. Harinarayan, A. Rajaraman and J. D. Ullman with lattice framework, proposed a greedy algorithm, BPUS (benefit per unit space) that attempts to maximize the benefit of the set of cuboids selected. Later, A.Shukla, P. M. Deshpande, and J. F. Naughton proposed PBS (Pick By Size) algorithm, which runs several orders of magnitude faster than BPUS. It uses a chunk based precomputation, which has a benefit based cost model for chunks.

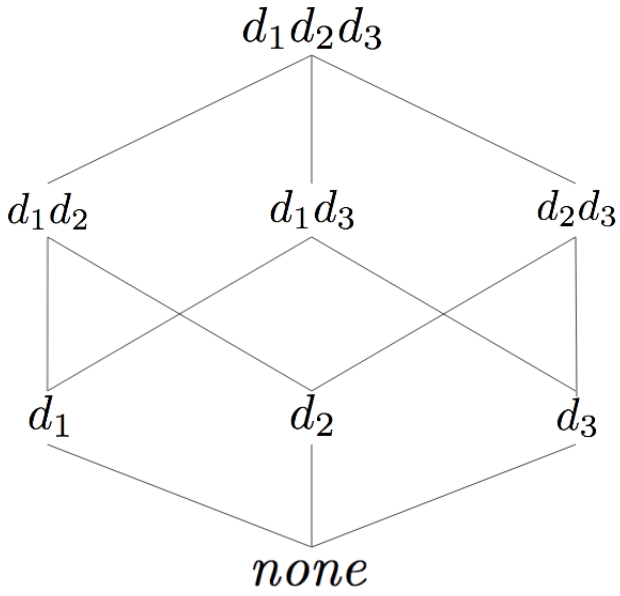


Figure 3.2: Lattice diagram of cuboid

3.2.3 Iceberg Cubes

Iceberg cubes is another partial materialization technique in data cubes. Min Fang et al. first proposed the concept of Iceberg cubes in their paper [26]. The base idea behind iceberg cube is, finding measures (aggregated values) above given threshold value, which produce small dataset (the tip of an iceberg) relative to a large amount of dataset (the iceberg). The following is given a typical iceberg cube with relational CUBE operation.

```
SELECT d1, d2, d3, SUM(M)
FROM R
GROUP BY d1, d2, d3, WITH CUBE
HAVING SUM(M) >= T
```

The work from Min Fang et al. [26] use a top-down approach in calculating Iceberg cubes. A bottom-up computation approached called BUC was later proposed by K. Beyer and R. Ramakrishnan for iceberg cube [13], which construct cube from the bottom of the lattice, and avoid calculation of large cuboids that does not meet the threshold value. J. Han et al. further improved the iceberg cube with H-Cubing method that uses Hypertree structure (H-Tree) [33].

Taking strengths from both BUC and H-Tree models, Dong Xin et al. proposed a novel approach called Star-cubing, which perform simultaneous aggregation on multiple dimensions and utilize iceberg conditions [77]. Later,

Z. Shao, J. Han, and D. Xin highlighted that none of the previous methods in calculating iceberg cube consider sparse cubes, and proposed MM-Cubing approach [60]. Their result shows MM-Cubing perform equally as BUC and Start-cubing for a uniformly distributed dataset, and outperform both when the data is skewed.

3.3 Data Cube compression

Another research approach in data cube materialization is to find a more compressed model for the data cube to reduce large access overhead. Most of the compressed technique consider archiving the model, which cause data nonqueriable. The research problem is to find techniques that compressed the data model while keeping the full queriability and random access to the measures.

3.3.1 QuantiCubes

P. Furtado and H. Madeira proposed a data cube compression strategy [27] called QuantiCubes, which uses a fixed-width compression coding to compress the data cube values. The result shows the compression technique achieve 2 to 5 times compression rate, and importantly decompression overhead is significantly low.

3.3.2 Condensed Cube

Wei Wang et al. proposed Condensed Cube [74] that reduce size of the data cubes and hence achieve better performance. Even-though condensed cube reduces the size, it is a fully computed cube and does not require further decompression or aggregation to response any OLAP queries. In general, tuples from different cuboids are shared if they are known being aggregated from the same set of tuples.

3.3.3 Quotient Cube

L. V. S. Lakshmanan, J. Pei, and J. Han proposed Quotient cube, which improves the semantic of the cube by partitioning the cube into a set of cells with similar behavior [47]. It is further extended in the paper [46], QC-tree, which solve most of the issues remained in Quotient cubes.

3.3.4 Graph Cube

Yannis Sismanis et al. proposed a graph-based approach, which consider a cube compression mechanism known as Dwarf cube [63]. It tries to solve space issue by

identifying prefix and suffix redundancies in the structure of the cube and factoring them out of the store. Dwarf is a directed acyclic graph (DAG) with one root, which has levels equal to the cube's dimensions.

J. C. Lima and C. M. Hirata proposed the Direct Acyclic Graph Cubing (MDAG-Cubing) model, which reduces cube computation effort and cube representation size [16].

3.4 Approximated Cube

The data cube approximation is another research direction, which builds on naive assumption that Analysts are not interested in an accurate value of the aggregation. Providing an approximate result is sufficient enough to make any important decisions and to identify any useful trends. The approximated cube find possibilities that reduce cube size and computation cost.

3.4.1 Quasi-Cubes

D. Barbara and M. Sullivan proposed Quasi-Cubes, as an approximation technique to reduce the storage cost of the cube [11]. It doesn't calculate cube measure's accurately. In this model, data cube is divided into region and use statistical model to describe each region.

3.4.2 Wavelet Decomposition

J. S. Vitter, M. Wang, and B. R. Iyer proposed an efficient technique that compact and accurate representation of data cube using multi-resolution wavelet decomposition [73].

Chapter 4

Distributed Data Cube Materialization

Early research direction in Data Cube Materialization considers data resides in a centralized data repository and utilizes resources in a single computation node. But, modern intensive datasets are no longer tolerate in a centralized Data Warehouse. The conventional ETL data processors that pull data from multiple Data Sources into target Data Warehouses do not scale with today's intensive data collections, Big Data. Thus, it requires a distributed computing approach for materializing data cubes. Fortunately, there are distributed computing frameworks like MapReduce and Resilient Distributed Datasets (RDD), which abstracts the underlying complexities of Distributed Computing and provides simplified API to build Distributed applications.

Several researchers have already proposed Data Cube Materialization algorithm using MapReduce. Among them, MRDataCube [48] algorithm considers optimization of MapReduce for Data Cube Materialization and shows better results. Apache Spark further simplifies Data Cube Materialization with its DataFrame API. It includes CUBE operator as part of the DataFrame API, which allows full Data Cube Materialization for given data dimensions.

4.1 Data Intensive Computing

The computational power has continuously increased during the last two decades. At the same time, the cost of computation and storage have decreased tremendously. This brought the capabilities of software components beyond its limits, but with the challenge of handling overwhelming data flows. The term Big Data was emerged to refer these massive volumes of the data collections with the new challenge of Data-intensive computing.

Big data is the data characterized by 3 attributes: volume, variety and velocity.
— IBM

The Big Data implies a technical problem of processing *varieties* of high *volume* data with different growing *velocities*. The Distributed Computing paradigm addresses this problem by sharing computing and storage capacities through a cluster of computer nodes. There are two main challenges in this approach;

1. Maintenance of cluster of nodes is difficult and expensive.
2. The design and implementation of distributed applications are challenging. It requires a consideration of more complex techniques in distributed system such as parallelization, fault-tolerance, data distribution, load balancing, etc

The new computation infrastructure paradigm, *Cloud Computing* addresses the first problem in distributed computing. It allows even smaller enterprises to step into the era of Big Data applications. Cloud Computing shifts the location of computing infrastructure to the Internet as a service, which reduces the cost and complexity of managing hardware and software resources [8].

The second problem in Distributed Computing is also already addressed in various Distributed System frameworks that abstract the underlying complexities of distributed systems and let application developers focus only on the application logic. The Distributed System frameworks such as Apache Hadoop [3], Apache Spark [6], Apache Storm[7], etc. provides a standard distributed programming model for applications to be executed on a cluster of computing nodes with the guarantee of scalability and fault tolerance. These Distributed Systems Frameworks simplify the problem of distributed computing under three main areas; 1) Distributed Storage, 2) Distributed Processing and 3) Distributed Resource Scheduling.

Distributed Storage Distributed Storage considers management of large-scale data on a network of computer nodes. The main problem it addresses is that modern intensive data outgrows the storage capacity of a single computer node. Thus data need partitioning across a cluster of computer nodes while ensuring the scalability and fault tolerance. Google File System (GFS) [28] is well established distribute storage mechanism today implemented by Google Inc. GFS is led with main design assumptions that the system is built from multiple commodity computing nodes that fail often. The system stores large files with large streaming reads and writes that append data to files. Taking its design essence, Apache Hadoop Distributed File System (HDFS) [35] publishes an open source version of Distributed File System that follows similar design assumptions as GFS.

Opposed to standard File Systems, Database Managment Systems (DBMS) provides more advanced features on a collection of data across multiple users. Relational Database Managment Systems (RDBMS) dominates the area of structured data management with ACID properties and declarative query language SQL. But RDBMS are not scalable over the growing dataset. Usually, RDBMS achieves the scalability with more expensive and complicated techniques such as Replications and Sharding. The new DBMS stream, NoSQL simplifies the distributed design strategy for achieving scalability by weakening some of the ACID properties. Google Big Table [17] and Apache HBase [4] are well know Distributed Database Management Systems today that do not guarantee high availability. The DBMS like DynamoDb [21] and Cassandra [45] are highly available Distributed DBMS but do not guarantee strong consistency.

Distributed Processing Distributed Data Processing involves applying data processing logic over a large-scale data collection that resides in a distributed storage system. Google's MapReduce [37], a distributed programming model first established the idea of taking data processing logic closed to the data. MapReduce gets benefits of the underlying distributed file system, GFS and operates parallelly over the distributed data blocks in the file system. Hadoop MapReduce [31] brings the same Distributed Processing Model for HDFS to the open source community.

But MapReduce has some drawbacks when it comes to iterative data processing models as it heavily involved local files in HDFS. Spark's RDD [79] overcomes this central issue by its in-memory distributed computing model. Apache Spark as a Distributed Data Processing model has various advantages over MapReduce. It extends with more operators and supports advanced data processing flows. Apache Spark is one of a widely attractive open source project today with a different contribution of sub-projects from the community. The subcomponents include Spark SQL, Machine Learning Library, Spark Streaming and GraphX.

Distributed Resource Scheduling Having a Distributed Data Processing mechanism operates on a Distributed Storage system solves a half of the problem for Distributed Applications. The next primary challenge is to allocate distributed resources (Memory, CPU) efficiently among distributed processes. Most of the early work in Distributed Application Framework like Apache Hadoop had its own resource scheduling mechanism but was more monolithic as there was no clear separation of data processing with resource scheduling, which brings a challenge of sharing resources with other Distributed Processing framework.

It brought the motivation for resource scheduling mechanism that allows many Data Processing Frameworks to run on a single cluster of nodes. The resource management system maximizes the utilization of resources among multiple distributed processes from different frameworks. Apache Mesos [36] and Apache YARN [71] are widely used Resource Managment systems today.

4.2 Big Data Analytics

Data Analytics is the major demand for modern large-scale data collections. The conventional data analytics architecture for OLAP explained in section 2.4 does not support current intensive data collections and their technologies. The data sources in modern online transactional systems use some NoSQL Distributes Data Storage mechanism like Cassandra, DynamoDB, etc. The traditional ETL processes that load data from multiple sources to a centralized Data Warehouse is no longer fit with modern distributed technology stacks. The Data Warehouse itself should be a distributed storage system. At the same time, the intensive data collections do not laid-back the response time on analytical queries. The Data Analysts and Decision Makers still expect near real-time query response time over any intensive data collections.

There is no particular standard architecture for advanced Big Data Analytics. But a stack of technologies exists today that define this architecture. SMACK is well-known technology stack used for Big Data Architectures [25]. SMACK stands for the following technologies and collectively desire functionality of each technologies determine a Big Data Architecture;

Spark: Data Processing Engine

Mesos: Resource Scheduler or Container

Akka: Distributed model for application development.

Cassandra: Distributed Storage

Kafka : The distributed message broker

The company Mesosphere in collaboration with Cisco coined the term SMACK and produced a product called Infinity which bundled all these technologies together. The favorable approach for using SMACK is to adopt only relevant technology stacks. Also, it is always convenient to replace any of the technologies in the stack with another that covers the same purpose and best suited the solution domain. Figure 4.1 taken from the book [25, p.12] represents SMACK at a glance.

Figure 4.1: SMACK Overview [25, p.12]

As it depicts in figure 4.1, there is no a data Extraction or Load phase. Instead, a distributed message broker is listen on data from diverse data sources. The source data then stream into a Distributed Processing system and uploaded into a Distributed storage systems.

4.3 Data Cube Materialization for Big Data

As explained in the previous chapter, Data Cube Materialization addresses two main challenges; Efficient Storage and Computation techniques. These challenges are even vital in modern intensive data collections. The Data Cubes would be immensely large and computationally even expensive for modern Big Data. Thus, Data Cube Materialization should adapt a distributed system approach. Fortunatly, various distributed system frameworks available today, which abstract the underlying complexities of distributed systems. Therefore, the challenges of efficient storage mechanism for data cube materialization can be adapted with a distributed data storage mechanism. In the same way, an efficient computation of data cube materialization can be achieved with a distributed data processing model.

4.3.1 Data Cube Materialization with MapReduce

MapReduce is widely used distributed programming model for various distributed applications. It has been one of research direction using MapReduce for Data Cube Materialization, which utilizes data resides in Distributed File System (DFS) and apply Data Cube Materialization with some MapReduce algorithm.

4.3.1.1 MapReduceMerge

Yuxiang W., Aibo S. and Junzhou L. proposed *MapReduceMerge* [75], a parallel cube construction model based on Google MapReduce and Google File System. They have extended standard Google MapReduce programming primitive as the following to support multiple related heterogeneous dataset in Data Cube materialization.

map : $(\text{k1}, \text{v1})_a \rightarrow [(k2, v2)]_a$
reduce : $(k2, [v2])_a \rightarrow (k2, [v3])_a$
merge : $((k2, [v3])_a, (k3, [v4])_b) \rightarrow [(k4, v5)]_c$

As it states above, the *reduce* function in MapReduceMerge produces a list of key/values, which is the input for the additional *merge* function. The merge function combines output from reduce function from different data collections and result a list of key/values belongs to target materialized dataset.

MapReduceMerge employs GFS to segment the data vertically according to the dimensional attributes. That is, a raw data set *abcm*, where *a*, *b* and *c* are dimensional fields, and *m* is a measure field, produces three files separately on GFS containing data as *am*, *bm* and *cm*. In this way, it allows to load only relevant data for processing and ignore unnecessary data loads.

4.3.1.2 MR-Cube

Arnab Nandi et al. propose *MR-Cube* [1] a MapReduce model for efficient Data Cube Materialization. As authors state, MR-Cube is the first comprehensive study on Data Cube Materialization for holistic measures. The algorithm makes use of the data partition based on cube lattice and divide the computation into pieces such that reducers deal with equally distributed small data groups. Their results shows MR-Cube algorithm efficiently distributes the computation workload and scale perfectly.

4.3.1.3 MRDataCube

Recently Suan L., Sunhwa J., and Jinho K. developed MRDataCube [48], another MapReduce algorithm for Data Cube Materialization. It highlights the existing MapReduce-based algorithm for Data Cube Materialization have fundamental problems related to the cost and limited capacity. As they argue, increasing number of nodes can bring some computational gain but with extra cost. They propose MRDataCube, a new MapReduce-based algorithm capable of efficient Data Cube Materialization of large datasets over the same computing resources.

They had carried out a long series of experiments on existing MapReduce-based data cube algorithm and found it is important for Data Cube Materialization

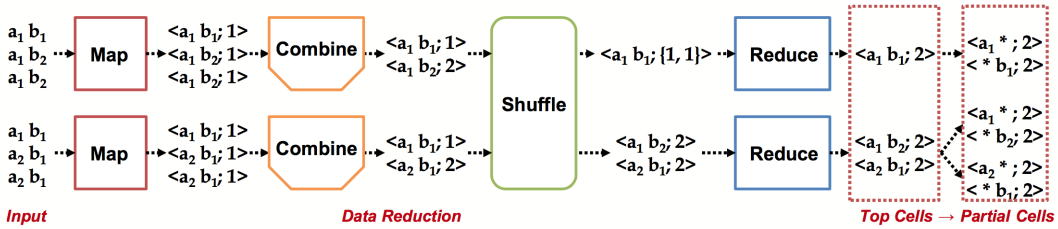


Figure 4.2: A sample data flow of MRSpread phase in MRDataCube [48]

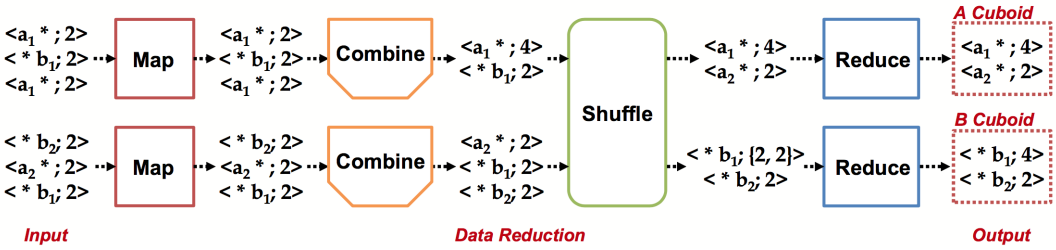


Figure 4.3: A sample data flow of MRAssemble phase in MRDataCube [48]

in MapReduce to use the distributed parallelism of MapReduce optimally. Their result shows MRDataCube outperforms all other existing existing MapReduce-based data cube algorithms.

MRDataCube algorithm mainly considers reducing output data collections from Map function and hence reduce data shuffling. They introduced two phases in MRDataCube to achieve this, MRSpread and MRAssemble. MRSpread emits all partial cells of the target cuboids, reducing all the overlapping dimensions with additional combiner function in MapReduce. Figure 4.2 (taken from original paper [48])shows an example data flow of MRSpread phase.

MRSpread generates all partial cells, which is input to the MRAssemble phase of MRDataCube. MRAssemble aggregates all the partial cells and generates all cuboids. Figure 4.3 (taken from original paper [48]) depicts the sample data flow for MRAssemble phase.

4.3.2 Data Cube Materialization with Apache Spark

Matei Zaharia et al. explains in their paper [79], Distributed Computing framework like MapReduce is inefficient for iterative algorithms and interactive data mining tools. Thus, promotes in-memory Distributed Data Abstraction, Resilient Distributed Dataset (RDD), which improves performance by an order of magnitude. Apache Spark [6] uses RDD as the underline distributed computing mechanism for various Distributed Sub-components such as Spark SQL, Stream Processing, Graph Processing and Machine Learning.

Unlike MapReduce, there is no any research carried out with RDD in Apache Spark on Data Cube Materialization. Instead Apache Spark introduces CUBE operator as part of its DataFrame API.

4.3.2.1 Spark DataFrame

Spark SQL [9] is one of main sub component in Apache Spark for Structured Data Processing. Unlike standard Spark RDD, Spark SQL allows distributed applications to leverage the benefits of declarative queries and optimized storage. Spark SQL enables its declarative query capabilities through its DataFrame API. In addition, Spark SQL includes *Catalyst*, a a highly extensible optimizer that perform extra optimization in data processing.

DataFrame is the central abstraction in Spark SQL, which represent a distributed collection of data rows with a homogeneous schema. It is conceptually equal to the Tables in relational databases and supports various relational operations with more optimized execution. DataFrames are constructable from external Distributed Data sources like HDFS, or from existing RDDs. Once constructed, the data collection abstracted in DataFrames are manageable with its relational operators.

4.3.2.2 DataFrame's CUBE Operator

In June 2015, Apache Spark version 1.4.0 included CUBE operator as part of the DataFrame API. It is same CUBE relational operator explained in section 3.1.1, which was introduced by Jim Gray et al. [30] that generates all possible GROUP BY for a given dataset. Figure 3.1 depicts the semantics usage of CUBE relational operator with a sample dataset and its result.

As a relational operator, CUBE is more convenient to be a part of Spark DataFrameAPI. It is just an extension to the existing GROUP BY operator from which CUBE operator can be determined. The following lists all supported method signatures of CUBE operator as in Spark version 2.2.0 of Java API.

```
public RelationalGroupedDataset
    cube(Column... cols)

public RelationalGroupedDataset
    cube(scala.collection.Seq<Column> cols)

public RelationalGroupedDataset
    cube(java.lang.String col1,
        scala.collection.Seq<java.lang.String> cols)
```

```
public RelationalGroupedDataset
    cube(java.lang.String col1,
        java.lang.String... cols)
```

In all cases above, Spark creates a multi-dimensional cube type of *RelationalGroupedDataset* for the current DataFrame using the specified columns. RelationalGroupedDataset abstracts a set of aggregation functions, which in return a new DataFrame.

Let us consider an application of CUBE operator with a sample DataFrame as given below. *df* is a DataFrame object with 4 columns *year*, *model*, *color*, and *sales*.

```
scala> df.select("year", "model", "color", "sales").show()
+----+-----+-----+-----+
|year|model|color|sales|
+----+-----+-----+-----+
|1991|Chevy| blue|   49|
|1991| Ford|  Red|   52|
|1990|Chevy|  Red|    5|
|1990|Chevy| blue|   52|
|1991| Ford|white|    9|
|1990|Chevy|white|   87|
|1990| Ford|  Red|   64|
|1992|Chevy|  Red|   31|
|1990| Ford| blue|   63|
|1992|Chevy| blue|   71|
|1991|Chevy|white|   95|
|1994| Ford| blue|   39|
|1990| Ford|white|   62|
|1991|Chevy|  Red|   54|
|1992|Chevy|white|   54|
+----+-----+-----+-----+
```

The following shows the usage of CUBE operator with two columns; *year* and *model*. And then application of SUM aggregation function for the column; *sales*.

```
scala> val fullDataCubes = df.cube("year", "model")
                             .agg(Map("sales" -> "sum"))
                             .cache()
```

```
scala> fullDataCubes.show()
+----+-----+----------+
|year|model|sum(sales)|
+----+-----+----------+
|1990| null|       333|
|null| null|       787|
|1991| null|       259|
|1992| Ford|        39|
|1992| null|       195|
|null|Chevy|       498|
|1990| Ford|       189|
|null| Ford|       289|
|1991|Chevy|       198|
|1992|Chevy|       156|
|1990|Chevy|       144|
|1991| Ford|        61|
+----+-----+----------+
```

As it can be seen in the result, CUBE operator generate the result for all possible data cubes for specified columns. The result contains data for four data cubes; Year-Model, Year, Model and None. The individual data cubes can be filtered out as the following.

```
# Data Cube: Year-Model
scala> fullDataCubes.filter("year is not null
                            and model is not null")
                    .show()
+----+-----+----------+
|year|model|sum(sales)|
+----+-----+----------+
|1992| Ford|        39|
|1990| Ford|       189|
|1991|Chevy|       198|
|1992|Chevy|       156|
|1990|Chevy|       144|
|1991| Ford|        61|
+----+-----+----------+
```

```
# Data Cube: Year
scala> fullDataCubes.filter("year is not null
                            and model is null")
                    .show()
+----+-----+----------+
|year|model|sum(sales)|
+----+-----+----------+
|1990| null|       333|
|1991| null|       259|
|1992| null|       195|
+----+-----+----------+

# Data Cube: Model
scala> fullDataCubes.filter("year is null
                            and model is not null")
                    .show()
+----+-----+----------+
|year|model|sum(sales)|
+----+-----+----------+
|null|Chevy|       498|
|null| Ford|       289|
+----+-----+----------+

# Data Cube: None Dimenions
scala> fullDataCubes.filter("year is null
                            and model is null")
                    .show()
+----+-----+----------+
|year|model|sum(sales)|
+----+-----+----------+
|null| null|       787|
+----+-----+----------+
```

In this way, CUBE operator of Spark DataFrame API provides a simplified approach for Full Data Cube Materialization for specified dimensional and measure fields. It hides all the data processing complexities over distributed storage systems and produces all possible data cubes in an optimized manner.

Chapter 5

Apache Spark vs Apache Hadoop in Data Cube Materialization

The thesis aims to contribute a standard OLAP Engine for modern Distributed Storage Systems such as Hadoop DFS, HBase, Cassandra, etc. The main challenge for such a framework is an efficient Distributed Data Cube Materialization approach. There are two main directions available today for Distributed Data Cube Materialization; MapReduce-based algorithms and CUBE operator of DataFrame API in Apache Spark.

With theoretical background of the two systems, Apache Spark has many advantages over MapReduce. But, their performance on Data Cube Materialization is still important to verify with a practical experiment. A quantitative research method with a set of experiments was designed to answer this question.

5.1 General Comparision of Apache Spark and Hadoop

We hypothesized the CUBE operator of DataFrame API in Apache Spark outperforms any MapReduce implementation for full data cube materialization. It is Apache Hadoop widely used distributed programming primitives for MapReduce and Distributed File System. Before dive into the main research problem, it is convenient to see a general comparison of Apache Spark and Hadoop.

There are various studies [59, 43, 38] available that compare Spark and Hadoop MapReduce. Raul Estrada and Isaac Ruiz in their book [25, pp. 12-14] provides a precise comparison of the two frameworks. Table 5.1 summarizes their work how Spark differs from MapReduce. As authors highlight Apache Spark includes a rich set of libraries for Machine Learning, Stream Processing, Graph Processing, Relational Data Processing. To achieve the same in Hadoop, it requires an integration of third party tools and libraries. The same is true with

Table 5.1: Apache Spark and MapReduce Comparison

	Apache Spark	**Hadoop MapReduce**
Written in	Scala/Akka	Java
APIs in	Java, Scala, Python, and R	Only Java
Storage Model	In Memory	In disk. Long latency in read/write from/to disk.
I/O Model	In memory without I/O	A lot of I/O activity over disk
Efficiency	Abstracts all the implementation to run it as efficiently as possible.	Programmers write complex code to optimize each MapReduce job.
Libraries	Adds libraries for machine learning, streaming, graph manipulation, and SQL.	Required third-party tools and libraries, which makes work complex.
Source Code	Scala programs have dozens of lines of code (LOC).	Java programs have hundreds of LOC.

materializing data cubes. DataFrame API in Apache Spark SQL already included with CUBE operator that perform data cube materialization. To achieve the same in MapReduce one has to implement a module from scratch or integrate such third-party library available.

Also, there is another advantage of using Apache Spark for data cube materialization. That is, Spark's broad range data processing support on various distributed storage systems such as inbuilt HDFS support, HBase connector, Cassandra connector, etc. In this way, Apache Spark allows data cube materialization on many distributed storage systems.

5.2 Performance Comparision of Apache Spark and Hadoop in Data Cube Materialization

There is no any research work available today that compare Apache Spark and MapReduce in materializing data cubes. It is one of the main contributions of this thesis that identify the best-distributed data processing approach for data cube materialization.

The main challenge here was to pick the best MapReduce algorithm for data cube materialization as several algorithms available from various research communities. Section 4.3.1 lists highly recognized MapReduce implementations in materializing data cubes. Among them, the recent work during year 2015-2016, *MRDataCube* by Suan Lee, Sunhwa Jo, and Jinho Kim states better

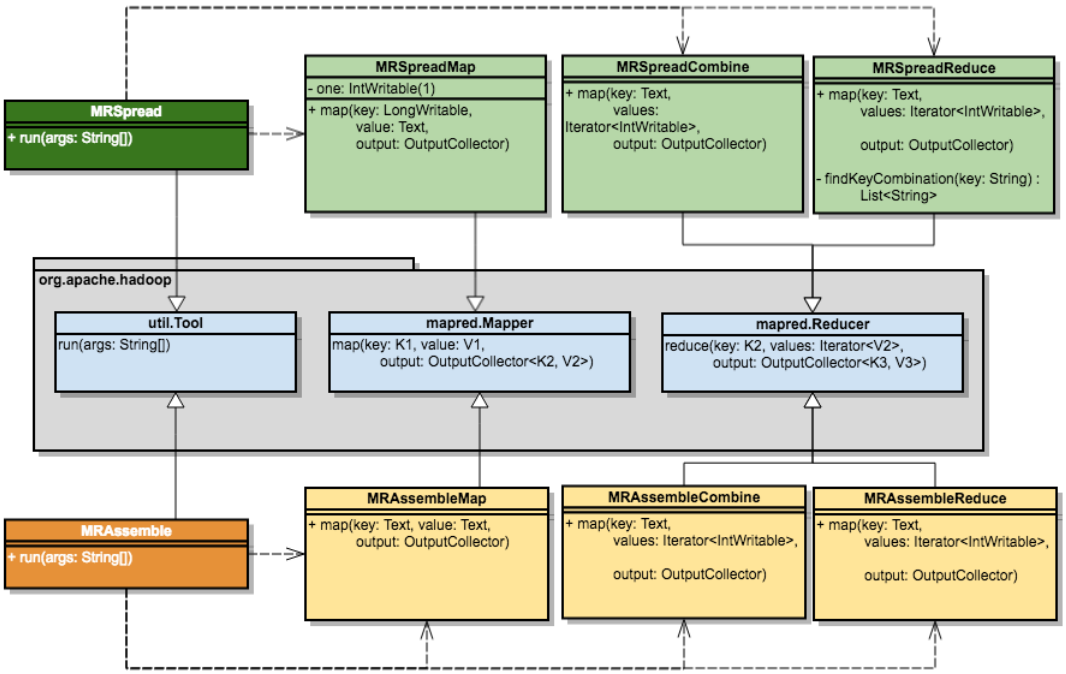


Figure 5.1: Testbed - MRDataCube Design Overview

performance compare to previous work in data cube materialization in their paper [48]. Therefore, the decision was to select MRDataCube algorithm to compare it with Data CUBE operator of DataFrame API in Apache Spark.

5.2.1 Testbed: Apache Spark and MapReduce implementation for Data Cube Materialization

Authors of MRDataCube do not share any implementation of the proposed algorithm. One of the tasks in setting up the testbed for the comparison was to implement MRDataCube algorithm. Figure 5.1 shows the design overview of the MRDataCube implementation.

As explained in section 4.3.1 MRDataCube algorithm composed of two phases; MRSpread and MRAssemble. The MRSpread phase determines the partial cuboids and makes them available for the MRAssemble phase. The MRAssemble phase calculates all the cuboids. We have further simplified the implementation of the testbed that MRSpread and MRAssemble determine only a single measure, COUNT. That is, for any given dataset it calculates the number of unique combination of the dimensional values.

The implementation for data cube materialization with Spark DataFrame's

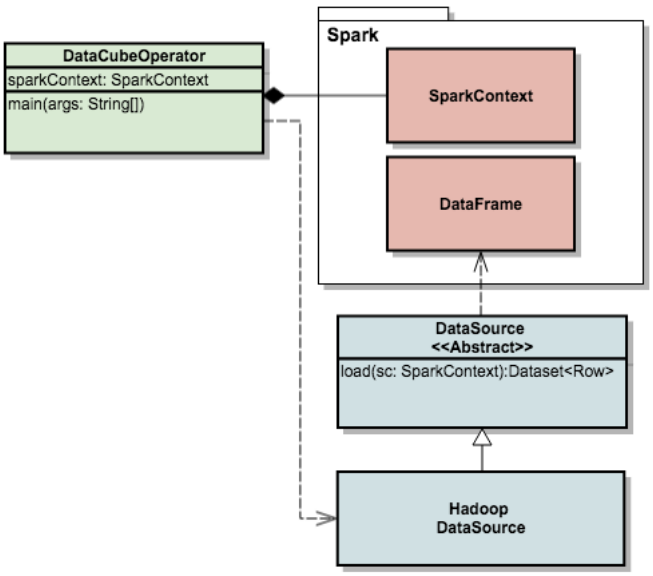


Figure 5.2: Testbed - Spark DataFrame's CUBE operator Design Overview

CUBE operator is straightforward. Figure 5.2 depicts the design overview of the implementation with Spark 's DataFrame. It simply loads DataFrame object from HDFS source and calls Cube methods with all columns fields and apply count operator. Both implementations calculate the elapsed time in performing Data Cube Materialization.

5.2.2 Testbed: Synthetic Large-scale Dataset

It states real world data are skewed and follows Zipf distributions. For example in natural languages, the frequency of any dictionary word is inversely proportional to its frequency rank, which is expressible in Zipf distributions [50] . The same applies with multidimensional datasets, and real world data cubes are not dense. Therefore, we generated a synthetic datasets according to Zipf distribution for the experiment. The Apache Commons Mathematics Library [65] provides the implementation for Zipf Distribution and Appendix A explains how the library was used to generate the synthetic dataset.

5.2.3 Testbed: Execution Environment

We selected AWS Cloud Service to set up the execution environment for the testbed as it provides excellent control and flexibility in arranging computing resources. Apache Spark and Hadoop clusters were setup on AWS EC2 instances

Table 5.2: Spark - Hadoop Cluster, The resource settings of *t2.large* EC2 instance

2 virtual CPU, 2.5 GHz, Intel Xeon Family
8 GiB Memory
200 GiB SSD

within same availability zone to minimize any network traffic. Appendix B, C, and D provides the complete guide for setting up Apache Spark and Hadoop clusters on AWS Environment. AWS EC2 instances were created from Amazon Linux AMI with the instance type t2.large that holds the resource settings given in table 5.2.

Figure 5.3 depicts the execution environment on AWS for all the test scenarios. As it illustrates in the figure all AWS EC2 instances were setup within eu-west-1 (Ireland) region and in same availability zone. In AWS, regions represent data centers and while availablity zones are identical to network racks. Therefore, the execution environment is equivalent to computer nodes in a single Data Center. Table 5.3 provides the base operating systems, software and application versions used in the Execution environment.

Table 5.3: Spark - Hadoop Cluster, OS, Software and Application versions

Amazon Linux AMI release 2017.03
Open JDK v1.8.0
Apache Hadoop 2.7.3
Apache Spark 2.1.0-bin-hadoop2.7

5.2.4 Testbed: Depended/Independent Variables and Test Scenarios

We considered the elapsed time for performing full data cube materialization as the dependent variable for the test scenarios. The elaspsed time is measured with the following independed variables which defined the test scenarios.

Number of Data Tuples The impact of data size on data cube materialization.

Number of Nodes in the cluster The impact of the cluster size on the performance of data cube materialization.

Number of Dimensions The impact of the number of dimensions in a dataset on the performance of data cube materialization.

The following test scenrios was defined according to above dependent and intependent variables.

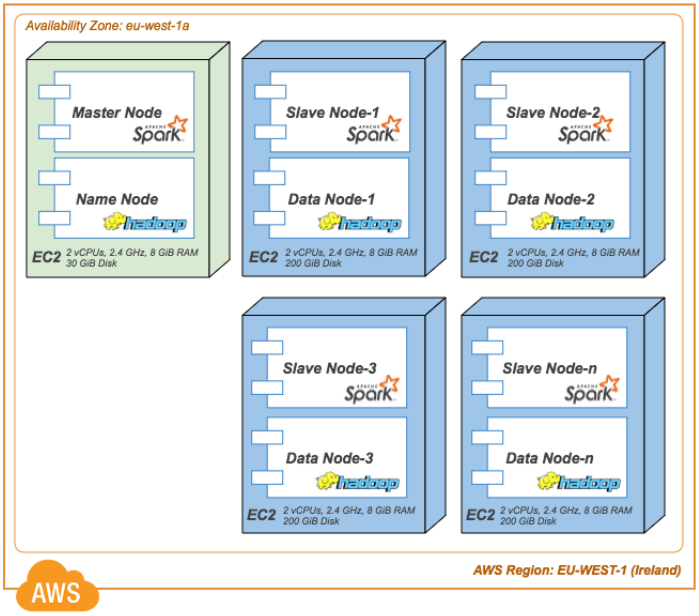


Figure 5.3: Testbed - Execution Enviornment on AWS

5.2.4.1 Data Cube Materialization over the Number of Data tuples

Seven datasets were generated starting from 1 million records to 7 million records with eight dimensions with maximum 64 cardinality level for each dimensional values. Full data cube materialization is carried out for each dataset with MapReduce MRDataCube algorithm and Spark CUBE operator separately on each cluster. The Spark cluster in this scenarios contained eight nodes for Slave processes and one separate node for the Master process. Similarly, Hadoop cluster contained 8 nodes for the Data Nodes and one separate node for the Name Node.

5.2.4.2 Data Cube Materialization over the Number of nodes in the Cluster

In this test scenario, a dataset with 4 million records with eight dimensions was used. And the elapsed time in data cube materialization was measured for different cluster sizes with Spark and Hadoop. The experiment was started with five nodes (One node for Spark Master and Hadoop Name Node, and rest of the nodes for Spark Slaves and Hadoop Data Nodes). Then the number of Spark Slaves and Hadoop Data Nodes were increased one by one until eight nodes.

5.2.4.3 Data Cube Materialization over the Number of Dimension

In this test case, six different datasets were generated with a different number of dimensions, from 3 to 8. The number of records was restricted to 1 million tuples. The experiment was carried out for the two systems with cluster size of five nodes; One node for Spark Master and Hadoop Name Node, and four nodes for Spark Slaves and Hadoop Data Nodes

5.3 Result and Analysis of the Test Scenarios

The elapsed time in data cube materialization for Spark CUBE operator and MRDataCube MapReduce implementation was collected for each test scenarios. Appendix E provides the complete result for each test cases.

5.3.1 Elapsed Time over the Number of tuples

Figure 5.4 illustrates the outcome of the test scenario that measures elapsed time in data cube materialization over different dataset sizes. As it clearly seen in the chart, Spark's CUBE operator has obtained four times lower elapsed time compare to MapReduce's MRDataCube for each dataset size. Also, MapReduce's MRDataCube shows high linear increasing trend compare to Spark's CUBE operator. That is, MapReduce would reach to the maximum resource level a node can handle much earlier than Spark's CUBE operator.

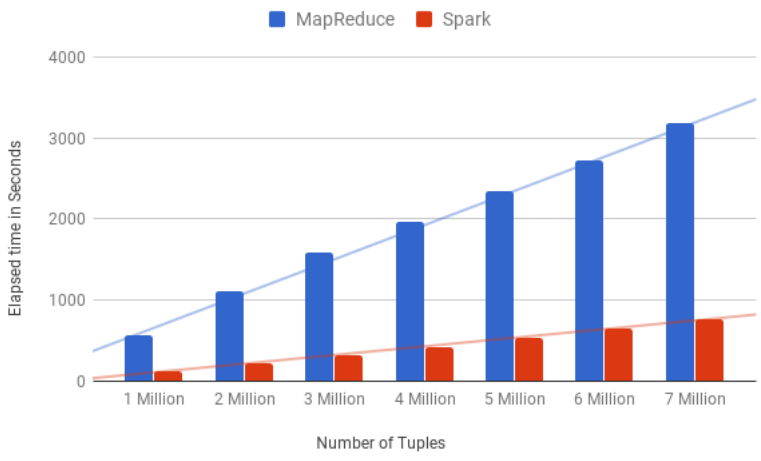


Figure 5.4: Elapsed time in Data Cube Materialization over the number of tuples in a dataset.

5.3.2 Elapsed Time over the Number of dimensions

Figure 5.5 illustrates the impact of number dimensions in a dataset on elapsed time of data cube materialization. As it presents, the number of dimensions has a significant impact on its performance for both systems. It follows the mathematical relationship in between the number of dimensions and its expected cuboids count. A data collection with n dimension causes a calculation of 2^n cuboids, which is exponential growth that same seen in the result.

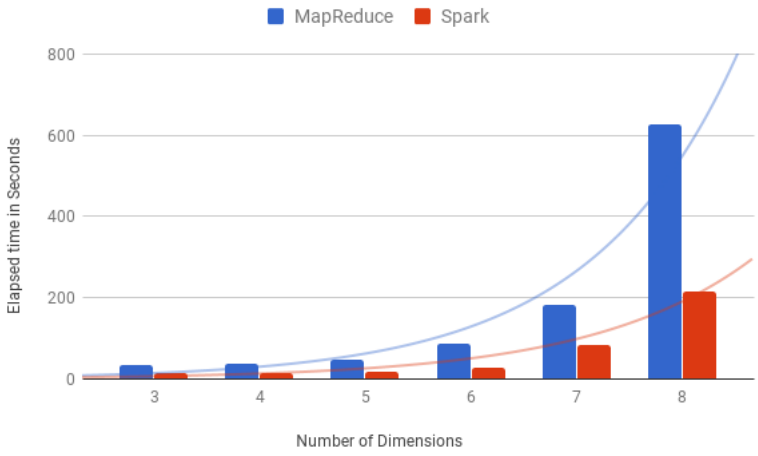


Figure 5.5: Elapsed time in Data Cube Materialization over the number of Dimensions

5.3.3 Elapsed Time over the Number of nodes

Figure 5.6 presents the result of the test scenario that measures elapsed time in data cube materialization over the cluster size of two systems. In both systems, increasing cluster size can achieve some improvement in the data cube materialization. But the decreasing trend of elapsed time is not linear. It's more polynomial that at some points, the number of nodes in the cluster would have no impact on the elapsed time. This follows distributed storage mechanism of the two systems. Both uses the data partition of HDFS, and file size determines the number blocks and hence the maximum number of data executors it needs.

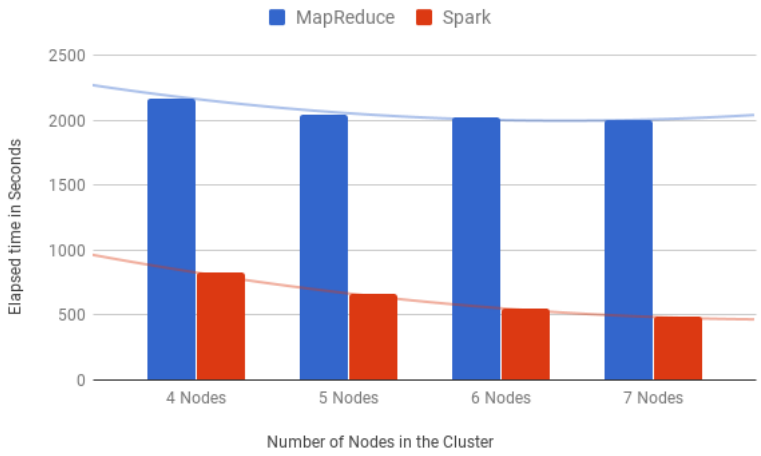


Figure 5.6: Elapsed time in Data Cube Materialization over the cluster size

Chapter 6

BigDataCube: A framework for Multidimensional Data Analysis with Apache Spark

Multidimensional Data Analysis or Online Analytical Processing (OLAP) is not discussed accordingly in current Big Data Analytics Paradigm. The main contribution of the thesis is *BigDataCube*, a framework for Online Analytical Processing over modern Distributed Storage Systems. We concluded in the previous chapter that Spark's CUBE operator in DataFrame API provides the most efficient distributed computing approach for Data Cube Materialization. The framework leverages Multidimensional Data Analysis with Spark's CUBE operator in DataFrame API.

In addition, the usage of Spark for materializing data cubes in the framework brings more additional advantages. Mainly, Spark supports data processing over various distributed storage systems, which make the framework capable for supporting Multidimensional Data Analysis over all Spark supported data sources. Also, Spark comes with other distributed components, such as MLib, GraphX, Streaming, etc., which makes the framework easily extended to provide more advanced data analytical capabilities.

6.1 BigDataCube, Distributed OLAP Engine over Apache Spark

BigDataCube is the main contribution of the thesis, a new framework based on Apache Spark that enables multidimensional data analysis over modern distributed storage systems. It is a distributed OLAP engine, which leverages Spark's DataFrame CUBE operator for Materizalizing data cubes. Figure 6.1

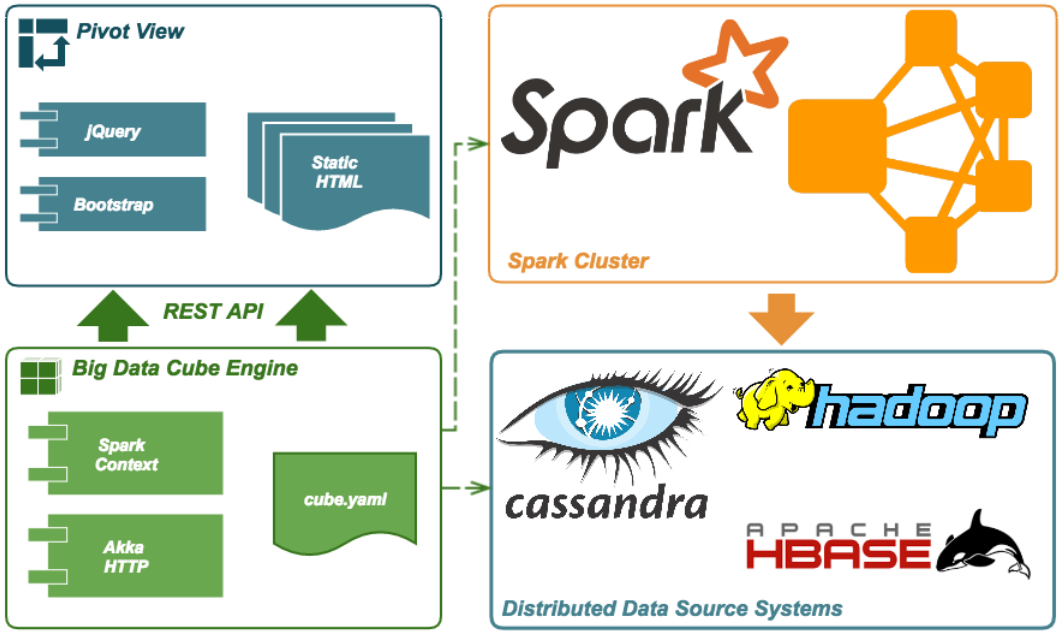


Figure 6.1: BigDataCube, System Overview

illustrates the system overview of the framework. As it depicts in the diagram, the framework composes of BigDataCube engine, which exposes a REST API for the front-end Data Analysis tool, Pivot View. BigDataCube engine is a standard Spark Application deployable on any Spark Cluster and configurable to define required Dimensions and Measures of the target Data Cube. But unlike standard Spark applications, once the BigDataCube engine is deployed it allows interactive access to the Spark Context via a RESTful API. The Spark Context holds the reference to DataFrames that represents Data Cubes. The front-end tool, Pivot View calls RESTful services to retrieve the data cubes and visualizes on Pivot tables.

BigDataCube engine consists of three main sub-modules; *conf/cube.yaml* configuration file, Akka HTTP endpoint, and Spark Context. At the deployment of BigDataCube engine, it first reads the configurations in *cube.yaml* and determine the Spark application settings, target data source, and dimension/measure fields of the Data Cubes. From data source configurations, it lets the relevant data source implementation to load the data as DataFrames. Then it applies the CUBE operator to materialize all possible Data Cubes for specified dimensional and measure fields in *cube.yaml*. Finally, BigDataCube engine makes the Data Cubes interactively accessible over the RESTful API via HTTP Akka library. All these execution steps occurred at the submission of BigDataCube engine on

Spark Cluster and when it completes all the generated Data Cubes are ready for interactive data analysis with the Pivot View.

6.1.1 Configuration Overview

BigDataCube engine reads its configurations in YAML file format. YAML is a human-readable and computationally powerful data serialization language [78]. YAML allows structured configuration level similar to XML, but it is more lightweight and simple. BigDataCube engine expects YAML configuration file at *conf/cube.yaml*. The following lists the main configurations options in the file.

```
# CUBE.YAML: Big Data Cube Configurations
name: SalesCube
port: 9090

# Spark Applications/Runtime/Security Properties
sparkProperties:
    spark.driver.cores: 1
    spark.driver.memory: 1g
    spark.executor.memory: 1g

# Data Source Properties
datasource:
    type: cassandra
    host: 10.0.1.211
    keyspace: analytics
    table: sales

# Dimensions for the Data Cube
dimensions:
    -
      name: Model
      field: model
    -
      name: Color
      field: color
    -
      name: Sales Date
      field: datetime
      hierarchy: DateHierarchy
```

```
# Measures for the Data Cube
measures:
    -
      name: Number of Sales
      field: sales
      aggregateType: sum
```

As it presents above, there are four main configuration options. Firstly, the application name and port that BigDataCube engine accessible over the RESTful service. Secondly, Spark application properties such as Memory, CPU, and Security. Thirdly, Data Source settings that Spark connects over. Finally, the Dimensions and Measures expected in the target Data Cube. A dimension configuration holds three properties; name, field, and hierarchy. *name* and *field* are mandatory properties in Dimension configurations. The *filed* represent the actual data filed name in the target Data Source, and the *name* gives a label to it, which is visible in the Pivot View. *hierarchy* is an optional property. The value for *hierarchy* is an implementation of *DimensionHierarchy* type in BigDataCube engine, which defines dimensional levels for specified field. For example, *DateHierarchy* type may introduce three additional dimensional levels as Year, Month and Data for the target cubes. In the sample configuration above, a value 2009-05-28 for datatime field results three additional fields as year: 2009, month: May, day: 28. BigDataCube engine allows any custom implementations for Dimensional hierarchies. A measure configuration also holds three properties; name, field and aggregateType. *name* and *field* properties are same as with dimension properties. *aggregateType* is a mandatory property which states the data aggregation function for the measure field. It allows all the aggregation functions that Spark supports such as SUM, AVG, MAX, MIN, etc.

6.1.2 API Overview

BigDataCube engine exposes RESTful services that are mainly used by the Pivot View front end tool. But the REST API is useful for any other Data Analysis tools. The following describes the REST API of BigDataCube with RAML, a RESTful API Modeling Language [57].

```
#%RAML 0.8
----------
title: BigDataCube API
baseUri:http://HOST:9090/datacube
version:v1
```

```
/dimensions:
    GET:
        description: Return all dimensions
        responses:
            200:
                body:
                    application/json
                    example:
                    [
                        {
                            "name" : "Model",
                            "field":"model",
                            "hierarchy":"ModelHierarchy"
                        }
                    ]

/measures:
    GET:
        description: Return all measures
        responses:
            200:
                body:
                    application/json
                    example:
                    [
                        {
                            "name" : "Sales",
                            "field":"sales",
                            "agregateType": "sum"
                        }

                    ]

/cuboids/{cubeId}:
    GET:
        description: Return given Data Cube with id 'cubeId'
        responses:
            200:
                body:
                    application/json
                    example:
```

```
[
    {
        "model" : "Ford",
        "color":"Black",
        "year":"1990",
        "sales": "200"
    },
    {
        "model" : "Ford",
        "color":"White",
        "year":"1990",
        "sales": "190"
    }
]
```

6.1.3 Design Overview

One of the design challenges with BigDataCube engine is to provide a way for interactive data analysis. Usually, Spark applications considered as batch data processing jobs that read data from one data source and write back to same or another storage. There is no a particular approach for interactive data analysis with Spark Applications unless the *Spark Shell*. But it is significant for BigDataCube engine to provide interactive access to underline Data Cubes over the RESTful services mentiond in previous section. BigDataCube engine uses Akka Http library [49] to achieve this feature. Akka Http provides a lightweight approach for exposing services over Http on top of Akka actor system. BigDataCube engine enables the Spark context for Akka actor system. That is, it allows accesses to the Data Cubes in Spark context over a REST API.

Figure 6.2 presents detail design overview of BigDataCube engine. As it shown in the diagram, the implementation is based on Akka HTTP and Spark Core/SQL libraries. *DataSource* provides the abstraction for many Spark supported Distributed data storage systems. *DataCubeService* abstracts the implementation logic for the services that it exposes as RESTful API via *DataCubeServiceResource*. DataCubeServer holds object references to DataSource, DataCubeService, etc. and provides the entry point to the applications.

DataCubeService provides the abstraction for implementation of various Data Cube Materialization Algorithm. As it depicts in the diagram, the main approach is full cube materialization with Spark Data Frame CUBE operator. But the design allow possibilities for partial cube materialization. Also,

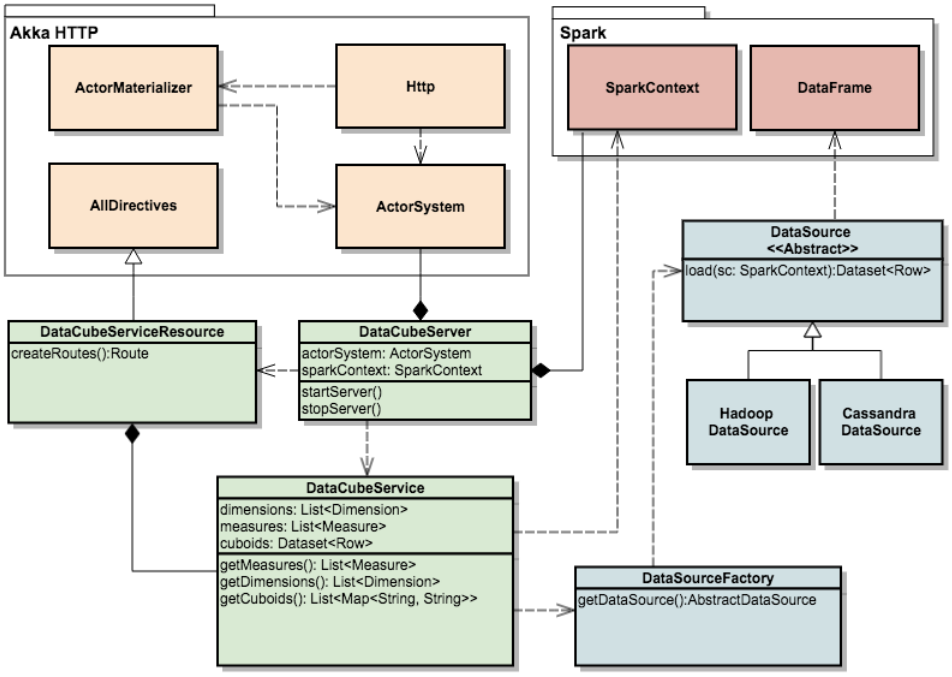


Figure 6.2: BigDataCube Engine, Design Overview

DataCubeService exposes several other services beneficial for the any Data Analysis tool.

DataSource provides the abstraction for many Spark supported distributed storage systems. For given configuration, the relevant implementation of the DataSource is initiated. Spark takes care of reading data in a distributed way and knows how data are partitioned across underlying distributed storage.

DataCubeServer combines all component as a Spark Application. It is the entry point for Spark master, and with the support of Akka HTTP service, it listens for HTTP requests defined in DataCubeServiceResource.

6.2 Pivot View, Visualization of Data Cubes

The next challenge is the visualization of the Data Cubes in a meaningful way that important insight of a data collection can be derived. Pivot View is part of the BigDataCube engine which brings front-end data analysis capabilities with Pivot tables. Pivot tables are quite known in spreadsheet applications like MS Excel. Also various other commercial and community tools available today [70, 55, 68].

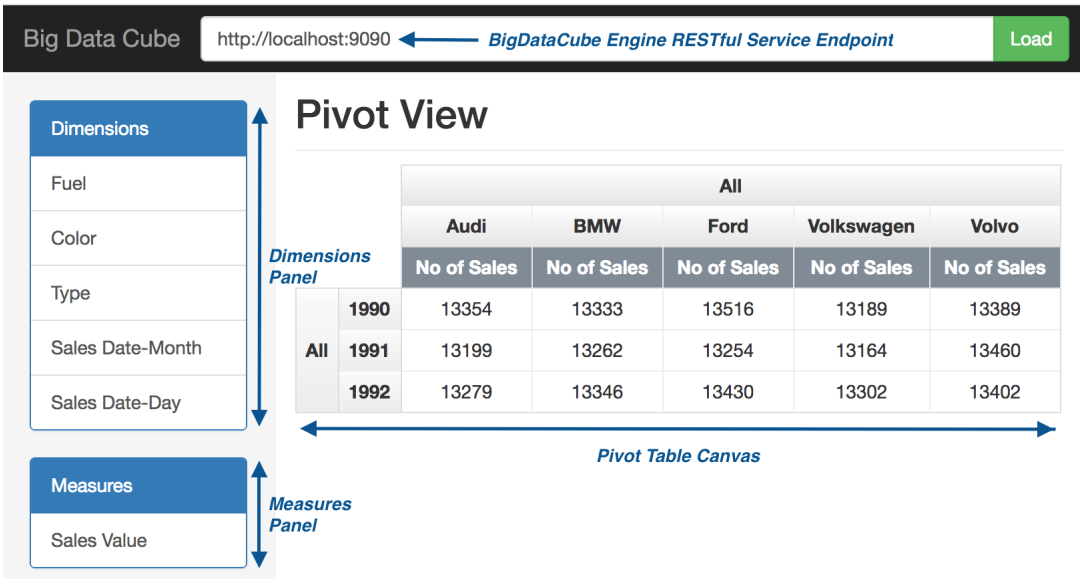
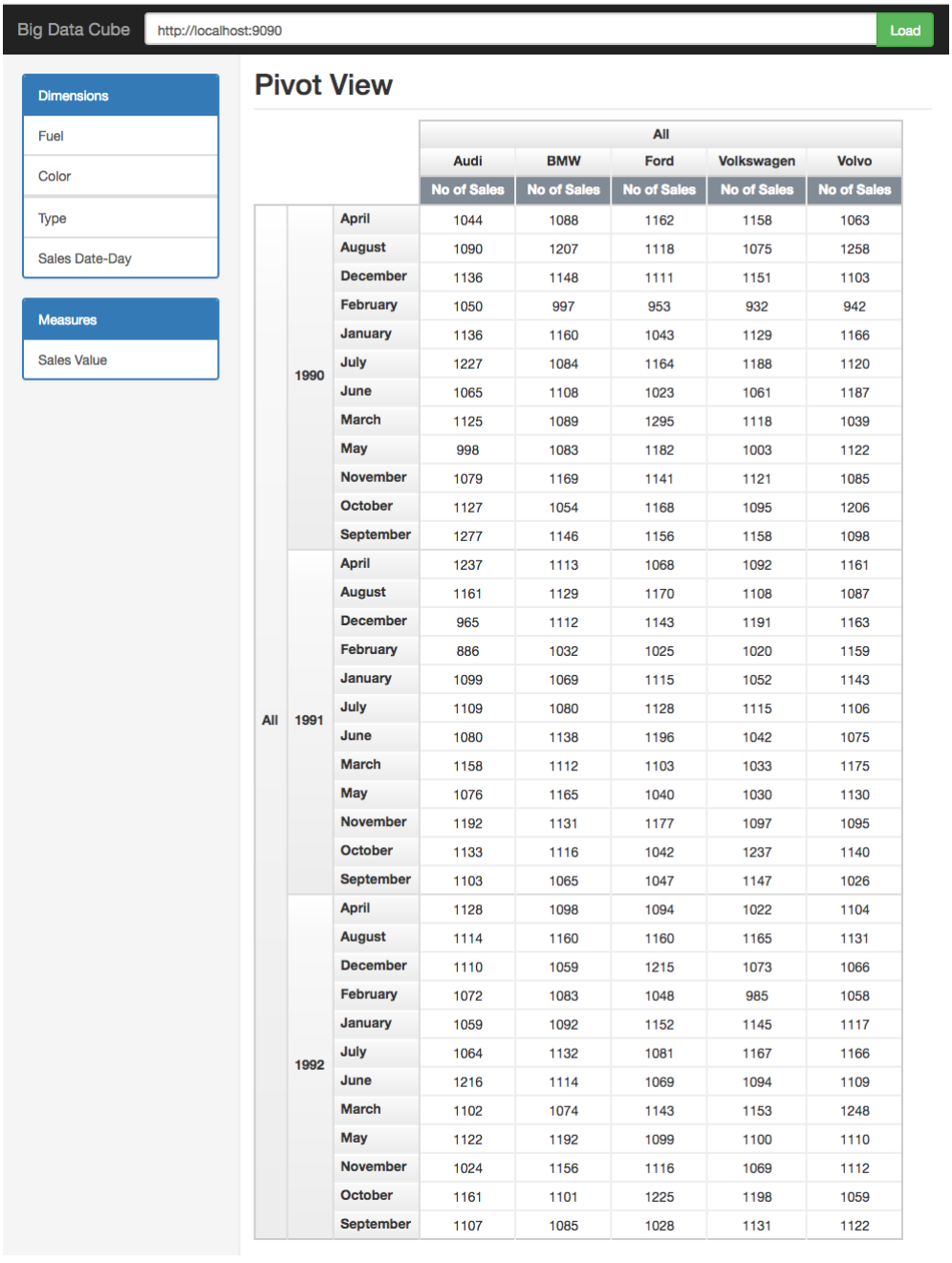


Figure 6.3: Pivot View UI Elements

Pivot View in BigDataCube engine is a static web javascript application. It is built with well-known Javascript libraries; jQuery [66] and Bootstap [14]. The pivot table is a modification of the original source jbPivot [70]. Figure 6.3 provides an overview the Pivot View UI and its elements. As it illustrates in the figure, Pivot View UI consists of three main UI elements; Dimensions Panel, Measures Panel and Pivot Table Canvas.

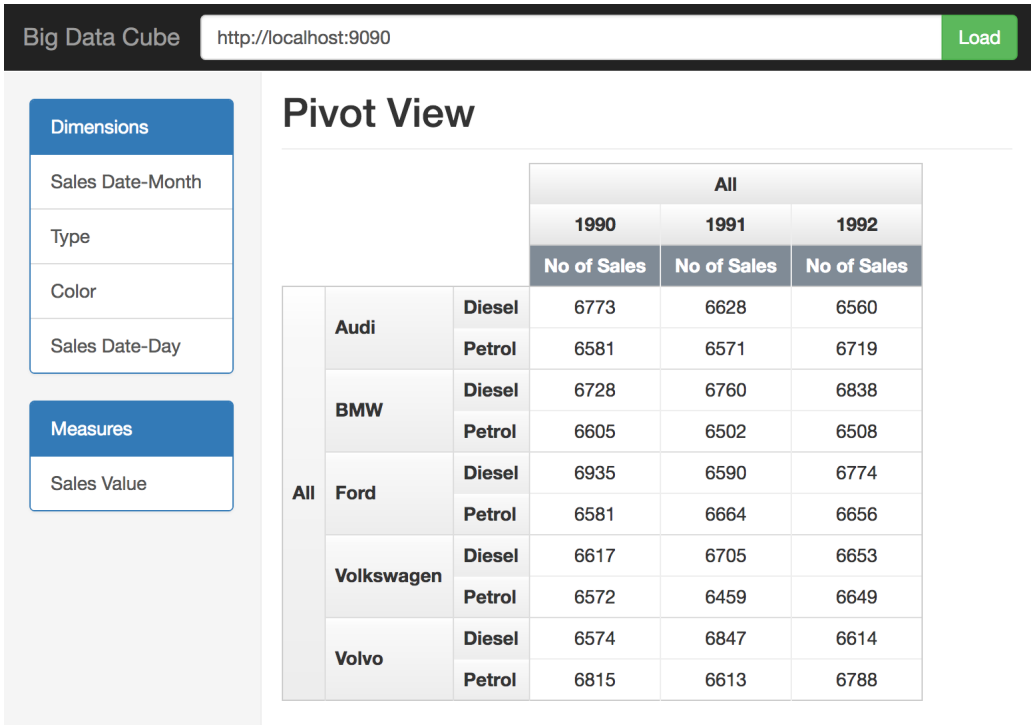
The Dimensions Panel is filled with the response from RESTful API call; $GET/dimensions$, which in returns all specified dimensions in the cube.yaml configuration file. In same way, $GET/measures$ RESTful API call from the Pivot View fills the Measures Panel. Initially, the Pivot Table Canvas is empty. The Dimension elements are drag and droppable into the Column or Row area of the Pivot Table. And the Measure elements are allowed within the body area of the Pivot Table. For each user actions (Drag and Drop of Dimensions and Measure elements), Pivot View generates a cube id and calls the RESTful service; $GET/cuboids/CUBEID$ on BigDataCube engine. BigDataCube engine can instantly respond to the Pivot View Request with all data cube records for given cube id, as it had already computed all the Data Cubes for specified dimensions and measures in cube.yaml at the startup. Upon the response from BigDataCube engine, the Pivot View simply display the records according to the arrangement of Dimensional elements in the Pivot Table.

Pivot View allows representation of various Cube operations within the UI. For example, Figure 6.4 shows how the *Number of Sales* measure can be further drilled down through the *Sales Date's* Month dimensional level.



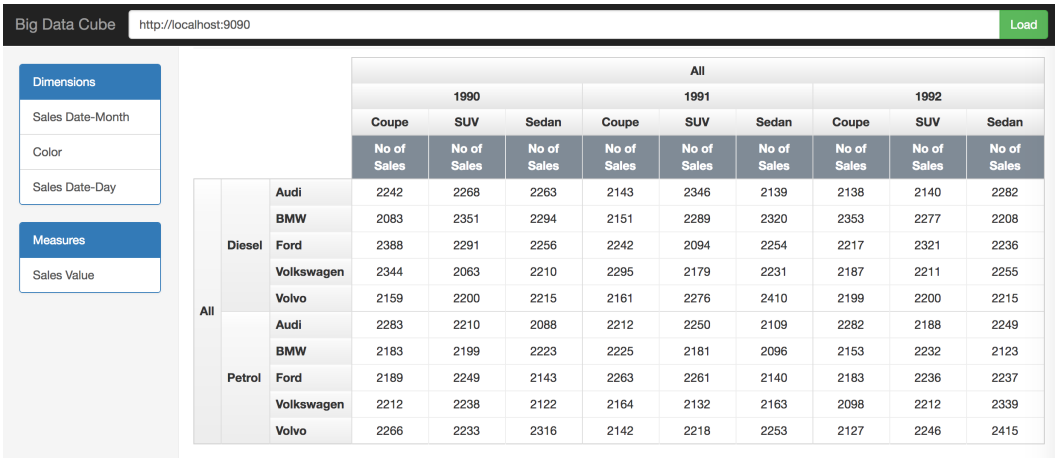
			All				
			Audi	BMW	Ford	Volkswagen	Volvo
			No of Sales	No of Sales	No of Sales	No of Sales	No of Sales
All	1990	April	1044	1088	1162	1158	1063
		August	1090	1207	1118	1075	1258
		December	1136	1148	1111	1151	1103
		February	1050	997	953	932	942
		January	1136	1160	1043	1129	1166
		July	1227	1084	1164	1188	1120
		June	1065	1108	1023	1061	1187
		March	1125	1089	1295	1118	1039
		May	998	1083	1182	1003	1122
		November	1079	1169	1141	1121	1085
		October	1127	1054	1168	1095	1206
		September	1277	1146	1156	1158	1098
	1991	April	1237	1113	1068	1092	1161
		August	1161	1129	1170	1108	1087
		December	965	1112	1143	1191	1163
		February	886	1032	1025	1020	1159
		January	1099	1069	1115	1052	1143
		July	1109	1080	1128	1115	1106
		June	1080	1138	1196	1042	1075
		March	1158	1112	1103	1033	1175
		May	1076	1165	1040	1030	1130
		November	1192	1131	1177	1097	1095
		October	1133	1116	1042	1237	1140
		September	1103	1065	1047	1147	1026
	1992	April	1128	1098	1094	1022	1104
		August	1114	1160	1160	1165	1131
		December	1110	1059	1215	1073	1066
		February	1072	1083	1048	985	1058
		January	1059	1092	1152	1145	1117
		July	1064	1132	1081	1167	1166
		June	1216	1114	1069	1094	1109
		March	1102	1074	1143	1153	1248
		May	1122	1192	1099	1100	1110
		November	1024	1156	1116	1069	1112
		October	1161	1101	1225	1198	1059
		September	1107	1085	1028	1131	1122

Figure 6.4: Pivot View - Drill Down through Date Dimension Level



			All		
			1990	1991	1992
			No of Sales	No of Sales	No of Sales
All	Audi	Diesel	6773	6628	6560
		Petrol	6581	6571	6719
	BMW	Diesel	6728	6760	6838
		Petrol	6605	6502	6508
	Ford	Diesel	6935	6590	6774
		Petrol	6581	6664	6656
	Volkswagen	Diesel	6617	6705	6653
		Petrol	6572	6459	6649
	Volvo	Diesel	6574	6847	6614
		Petrol	6815	6613	6788

Figure 6.5: Pivot View - Drill Down through Fuel Dimension



			All								
			1990			1991			1992		
			Coupe	SUV	Sedan	Coupe	SUV	Sedan	Coupe	SUV	Sedan
			No of Sales	No of Sales	No of Sales	No of Sales	No of Sales	No of Sales	No of Sales	No of Sales	No of Sales
All	Diesel	Audi	2242	2268	2263	2143	2346	2139	2138	2140	2282
		BMW	2083	2351	2294	2151	2289	2320	2353	2277	2208
		Ford	2388	2291	2256	2242	2094	2254	2217	2321	2236
		Volkswagen	2344	2063	2210	2295	2179	2231	2187	2211	2255
		Volvo	2159	2200	2215	2161	2276	2410	2199	2200	2215
	Petrol	Audi	2283	2210	2088	2212	2250	2109	2282	2188	2249
		BMW	2183	2199	2223	2225	2181	2096	2153	2232	2123
		Ford	2189	2249	2143	2263	2261	2140	2183	2236	2237
		Volkswagen	2212	2238	2122	2164	2132	2163	2098	2212	2339
		Volvo	2266	2233	2316	2142	2218	2253	2127	2246	2415

Figure 6.6: Pivot View with 4-Dimensions

Figure 6.5 is an example of Drill Down with a new dimension. As it illustrates in the figure, the Number of Sales now further breaks down with additional Fuel dimension. Also, Pivot View support visualization of Data Cubes with any number of dimensions. Figure 6.6 is an example of Pivot View for a 4-dimensional cube. In this way, Pivot View provides the Data Cube visualization capabilities and helps to gain important insight of a data collection.

Chapter 7

Case Study: BigDataCube over Apache Cassandra

Digital River World Payment is a service provider for global online payments. The standardized payment gateway allows integration of many merchants into the Payment Platform and routes their transaction over various Payment Networks. The payment platform consists of various service components, but a Data Analytical module is not yet well defined. In the thesis, BigDataCube framework is proposed with relevant architecture changes that bring Multidimensional Data Analytics to the payment platform.

Cassandra is the underlying database technology that payment platform uses to support high transactional throughput. Thus, it is the main data source for analytics. The thesis discusses an architectural approach how distributed data in Cassandra is used with BigDataCube framework with minimum impact on the transactions in payment platform. The result of the simple experiment shows the proposed architecture with BigDataCube framework is scalable in the long term.

7.1 Digital River Payment Gateway

Digital River World Payments is a service provider for global online payments. The online payment solutions cover various payments methods, including different card schemes and payments networks. It values integrated merchants with the full payment transaction lifecycle while providing add-on features such as fraud detection, consolidated reports, reconciliation, and back-office integration. The payment gateway for supporting a significant amount of transactions with add-on features in Digital River follows a standard product suite with well-determined technology platform.

The technology platform identifies two primary sub-systems that build up

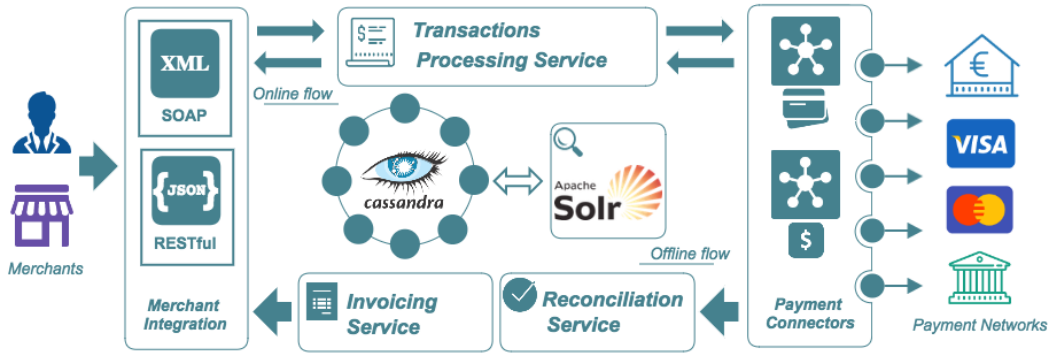


Figure 7.1: Digital River Payment Gateway - System Overview

the Payment Gateway. That is Online Payment Transaction Processing Systems and Offline Settlements Systems. Figure 7.1 depicts the overview of the system components and their base technologies. The primary platform services include Merchant Integration, Transaction Processing Service, Payment Connectors, Reconciliation Service and Invoicing Service.

Merchant Integration: The Merchant Integration System enables a simplified way to Merchants to integrate Payment Services and other facilities with their systems. It mainly exposes REST and SOAP API, with secure tokenize options.

Payment Transaction Processing Service: Payment Transaction Processing Service is the heart of the Payment Gateway. It processes payment transaction requests with various validation rules, routes over the Payment Connectors to the right destination, and send the response back to the merchant. All these steps are required to execute in a fraction of a second while supporting multiple transactions for many users.

Payment Connectors: Payment Connector System is an integration hub for connecting various payment providers, payment methods and financial institutions like banks. It is responsible for providing Transaction Processing System with proper payment connection to the external system.

Reconciliation System Reconciliation Systems are executed over offline payment connectors and register all the transaction settlements in batches, usually on a daily basis.

Invoicing System The invoicing system takes the settlements and tally with transactions and charges Merchants with auto-generated invoices.

All these systems are implemented on the same technology stack. As it depicts in the diagram among other propriety technologies, the main component is Cassandra, which provides high throughput on writing transactions. Solr integrated with Cassandra provides multi-index search options for all the ecosystems in the platform.

Cassandra: Cassandra is a distributed storage system for large-scale structured data that resides on a cluster of commodity servers [45]. It provides a distributed key-value store with eventually consistency model. It scales incrementally with the number of nodes in the cluster, and benefit from dynamic partitioning mechanism over consistent hashing.

Solr Solr is a distributed indexing system that provides a reliable search functionality. The data in Cassandra is sync with Solr and provides a multi-index search capability for data loaded from Cassandra.

7.2 The Missing Component

As it depicts in Figure 7.1, all the payment gateway services are running on top Cassandra. One of next level service component Digital River is looking for, Data Analytic capabilities that can synthesize Payment Transactions. Each service components own its keyspace and tables in Cassandra from which valuable insight of the data can be derived. For example;

1. The currency exchange rate in transactions in online flow can differ from the rate at the settlements. With the combination of payment transactions with their settlements details, Business Analyst can see if any potential profit/loss makes over the exchange rate.

2. Transactions per Merchant, Business Domain, Country, etc. allows the Business team to focus on the particular market.

The challenge in providing such analytic query capabilities is to bring another technology component that runs on Cassandra. Cassandra is a distributed data source. The traditional approach of extracting data into a centralized location might not scale with the growth of the dataset. BigDataCube engine explained in the previous chapter with Apache Cassandra would bring data analytic capabilities to Digital River Payment Platform. Figure 7.2 shows how the new component resides in the platform.

There are two main challenges in using BigDataCube framework in the Payment Platform. Firstly, setup or extraction of Cassandra data into an Analytical Data source, which minimizes the impact on online transaction system. Secondly,

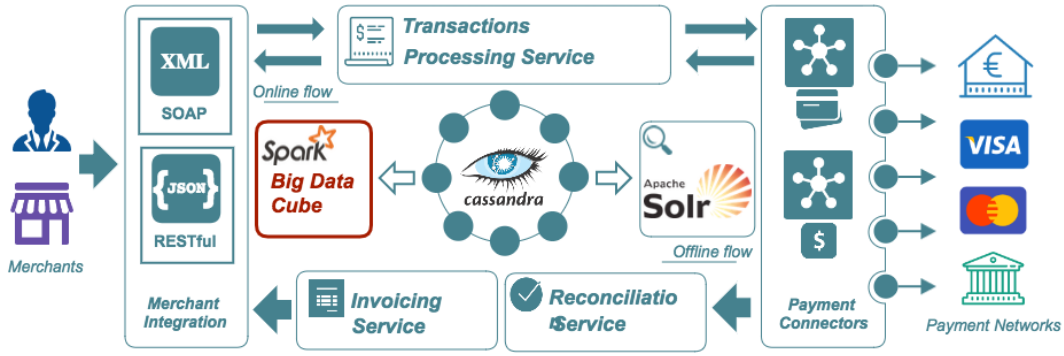


Figure 7.2: Digital River Payment Gateway - System Overview with BigDataCube Engine

the setup of Spark cluster that takes data processing jobs close to the Cassandra data. In other word bring computations close to the Cassandra data. The two main challenges in taking BigDataCube framework into the Payment Platform can be simply achieved with existing features and supported tools in Cassandra and Spark;

7.2.1 No E-TL Data Warehouse

The well-known design strategy in Data Analytical system is to bring Data into a separate system to minimize any impact on the transactional system upon analytical queries. Also, especially in the relational database area, the transaction system are optimized for the online transactions, data model and physical model are not suitable for analytical queries. The same applies for Payment Platform that runs on Cassandra. The analytical queries should not impact the online transactions. Also, the data model in the transactional system would not fulfill all analytical needs.

In the traditional approach, data is first extracted, transformed and uploaded into a separate data warehouse. The same applies to distributed storage, but streaming data from one cluster to another may not be practical nor scalable. The simplest approach is to replicate entire Cassandra ring, preferably in a separate data center (logical or physical) as depicts in figure 7.3. In this way, with minimum effort, the source data can be replicated into a separate Cassandra Cluster. Unlike traditional ETL procedures, the data is not pipelined through a set of procedures. Instead, any data transformations are applied to data partitions and the result is saved back in data partitions over a cluster of nodes.

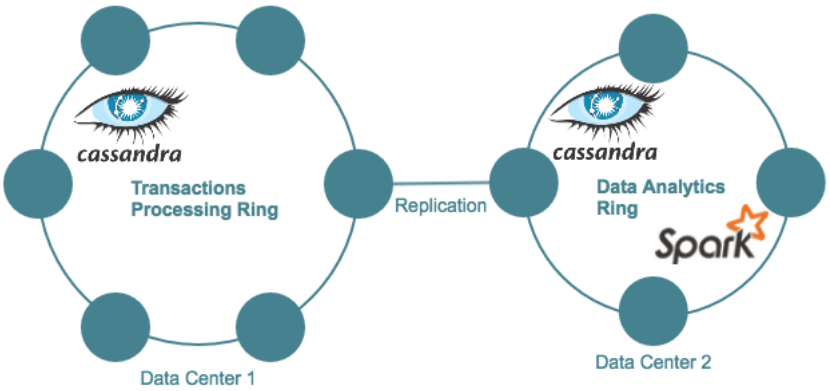


Figure 7.3: Cassandra Replication for Analytic Ring

7.2.2 Data Local Distributed Processing

The next challenge is how to bring computation close to Cassandra nodes. Fortunately, Datastax's Spark Cassandra Connector provides the abstraction for this. Like Spark works with Hadoop Distributed File system, where Spark executor assigned for each data node, Spark-Cassandra connector provides the similar capabilities. Appendix G gives detail overview Spark-Cassandra connector and how Spark brings data processing close to the data in Cassandra cluster.

7.3 BigDataCube over Cassandra

To evaluate Spark Data Processing over Cassandra with BigDataCube framework, a prototype of proposed Data Analytics architecture was setup on AWS. Also, a simple experiment was carried out to check the scalability of the platform and its performance.

7.3.1 Data Analytics Platform for Digital River

Figure 7.4 depicts a prototype of the system setup for target Data Analytical Platform. As it represents in the diagram, it is setup on AWS, the Cassandra cluster in a virtual private subnet, which assumes the transactional data are replicated using replication strategies in Cassandra. Spark Master is setup on a EC2 node in a public subnet, whereas all Spark Slaves are setup on same EC2 nodes that Cassandra nodes are running.

As it depicts in the diagram, instance of multiple BigDataCube engines are deployable on the Spark Cluster. Each BigDataCube engines may stand for

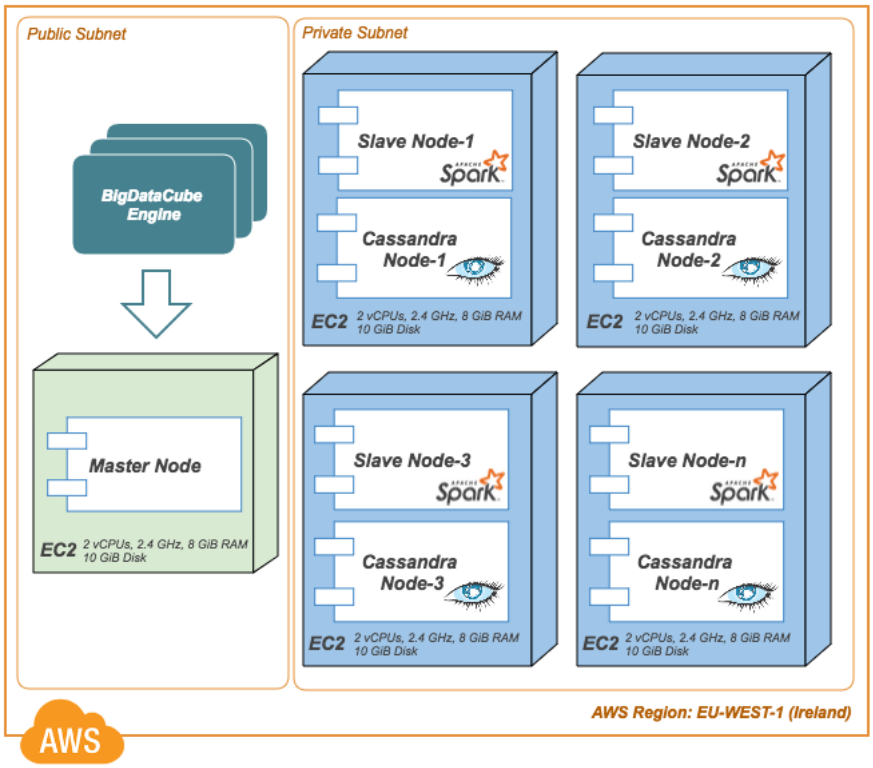


Figure 7.4: A Protoype Setup - Data Analytic Paltform for Digital River

different Data Cube that works with Cassandra tables. Both Cassandra and Spark are scalable over the number of Nodes, thus provides a scalable Data Analytical Platform for Digital River.

7.3.2 Scalability of the Data Analytic Platform

A simple experiment was to setup to verify the scalability of the platform. Spark-Cassandra Cluster on EC2 nodes with the settings table 7.1 - 7.2 was setup and measured the Data Cube Materialization time while changing the cluster size. A dataset with six dimensions and 10 million records was used on Cassandra-Spark.

Figure 7.5 represents the result of the experiment, where elapsed time of the Data Cube Materialization was measured increasing the cluster size of Spark-Cassandra from four to eight. From result we can conclude Data Cube Materialization over Cassandra also shows the similar behavior as it was with HDFS. For a larger dataset, increasing cluster size will always benifits with better performance.

Table 7.1: Spark - Cassandra Cluster, The resource settings of *t2.large* EC2 instance

2 virtual CPU, 2.5 GHz, Intel Xeon Family
8 GiB Memory
10 GiB SSD

Table 7.2: Spark - Cassandra Cluster, OS, Software and Application versions

Amazon Linux AMI release 2017.03
Open JDK v1.8.0
Apache Cassandra 3.10
Apache Spark 2.1.0-bin-hadoop2.7

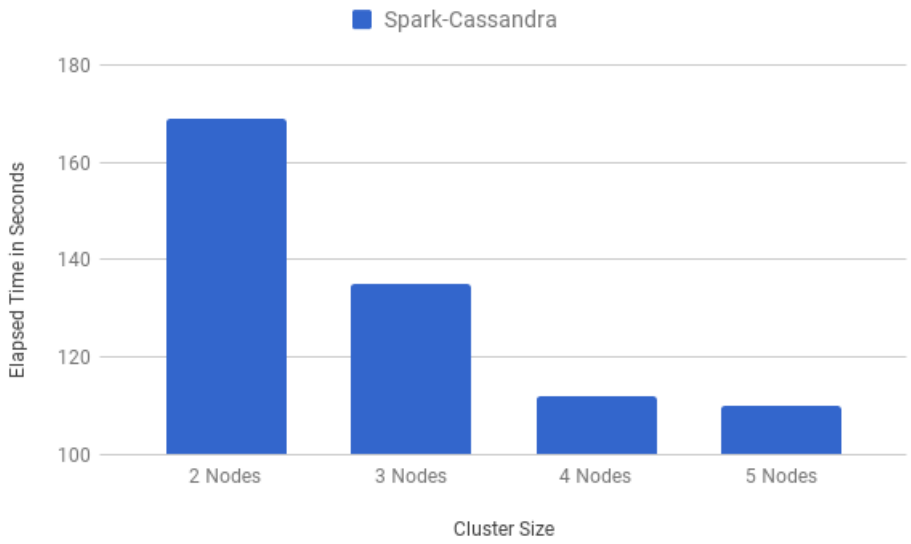


Figure 7.5: Elapsed time in Data Cube Materialization over the Cassandra cluster size

Chapter 8

Conclusion and Future Work

The thesis motivates the subject area of Multidimensional Data Analysis, which is an important division of Data Analytic Paradigm that helps in discovering valuable insight of data collections. Multidimensional Data Analysis considers all possible relationships among data attributes in a dataset, and the Data Cube abstraction provides a convenient Analytical Model to achieve it. In essence, many useful analytical queries are signified in Data Cubes and its operations. The thesis brings a detailed background of Data Cube notion and its importance in OLAP that facilitates on-line responses to analytical queries.

8.1 Data Cube Materialization is Important but Challenging

The primary possible technique for OLAP to provide online responses to analytical queries is to materialize all possible Data Cubes for a given dataset. That is, OLAP directs analytical queries to the relevant materialized Data Cube and response instantly without performing any data aggregations at runtime. But Data Cube Materialization is challenging in both storage and computation. An n dimensions dataset courses generation of 2^n number of data cubes. Also, the size of each Data Cube depends on cardinality level of each dimension.

The thesis provides a detail theoretical study of early research works in data cube materialization, which considers data resides in a centralized Data Warehouse system. Two main directions of data cube materialization seen in early research works; full and partial data cube materialization. CUBE operator initiates the base for most of the full data cube materialization approaches, while Lattice Framework receives the attention from partial data cube materialization direction. Among others, Data Cube Compression, Approximate Cubes, and Parallel Data Cube are recognizable methods in materializing data cubes.

The thesis raises the argument that traditional data cube materialization approaches are no longer applicable with modern intensive data collections which usually resides in a cluster of computer nodes. Thus, it requires a distributed data cube materialization method. With a theoretical study in Data Intensive Computing, it is apparent that various distributed computing modules have already abstract the complexities of Distributed Storage, Processing, and Resource Scheduling. Two leading such distributed processing systems are MapReduce and Spark. Both supports distributed data processing over Hadoop Distributed File System, and resource scheduling over YARN, Mesos, etc. Various studies already considered using MapReduce for data cube materialization. Among them, MRDataCube provides better results. Apache Spark also recently introduce CUBE operator as part of its DataFrame API, which performs full data cube materialization. The thesis raises the research problem; what is best-distributed computing approach for data cube materialization among Spark and MapReduce.

8.2 Apache Spark vs Hadoop MapReduce

RDD in Apache Spark emerged as an alternative for MapReduce, which lacks iterative data processing. With theoretical background of Apache Spark, it is apparent that Spark has more advantages over Hadoop MapReduce. To answer the research problem, the best-distributed computing approach for data cube materialization, the thesis conducts a testbed with the hypothesis that Spark provides the best-distributed computing model for data cube materialization. The testbed consists of two implementations, MRDataCube algorithm with Hadoop MapReduce and Spark's CUBE operator that both perform data cube materialization with a simple measure COUNT for specified dimensional attributes in a dataset. Three test cases determine elapsed time in materializing data cubes, against the cluster size, the number of records and the number of dimensions. The testbed is set up on AWS EC2 instances, and the results show similar trend from both systems but Spark with comparatively low elapsed time in data cube materialization.

8.3 BigDataCube, Distributed OLAP Engine

The next research problem the thesis addresses how OLAP allows multidimensional data analysis for the modern distributed systems. Kylin is the only available distributed OLAP engine today. But it does not follow data cube materialization. Instead, cube abstraction is enabled over Hive tables, which convert analytical queries to HQL. Therefore, an OLAP engine that employs data cube materialization

would lead to a better OLAP engine. The main contribution of the thesis, a distributed OLAP engine, BigDataCube, which base on Spark CUBE operator. It is a standard Spark Application, which is deployable on Spark Cluster. Once deployed, it performs full data cube materialization on configured distributed storage system with specified dimensional and measures fields. The result of the data cube materialization is a set of Data Cubes of Spark DataFrame type. BigDataCube exposes RESTful services to access each Data Cubes. BigDataCube has several advantages as it builds upon Spark. Mainly, Spark supports many distributed storage systems, which enable Data Cube Materialization over these systems. Also, the possibility to integrate it with other Spark modules, such as Machine Learning, Graph Processing, Stream Processing, etc. and build more advanced Data Analytics applications. In addition, BigDataCube considers the visualization of Data Cubes in Pivot views. It provides an interactive Pivot table (Javascript library), which calls RESTful services on BigDataCube engine and visualizes the resulted Data Cubes.

8.4 Use case for BigDataCube

The thesis is carried out at Digital River, a service provider for online payments. The online payment platform consists of various subsystems, each of which implemented in the same technology stack. Cassandra is the base for all the subsystems, which allows high write throughput on only payments transactions. The payment platform was looking for next level Data Analytical Subsystem.

We proposed a Data Analytic Architecture for their online Payment Gateway platform, which engages BigDataCube for Multidimensional Data Analysis. We set up a prototype of the proposed architecture on AWS and with an experiment that proves the scalability of the proposed solution.

8.5 Delimitation and Limitation

To answer the research problem, the best-distributed computing approach for data cube materialization, the testbed was setup with sufficient computer resources for both distributed systems; MapReduce and Spark. The experiments did not consider edge cases like behavior upon limited resources. That is, MapReduce and Spark performance in materializing data cube with limited disk and memory spaces. Among various MapReduce models, MRDataCube algorithm was chosen to compare with Spark DataFrame in materializing data cubes as the authors of MRDataCube clearly states it outperforms previous works in MapReduce for materializing data cubes. Also, the cardinality level of dimensions and data file

size was not considered as independent variables in measuring elapsed time for materializing data cubes. It is obvious the two cases covered in the test case with the independent variable, Number of records.

BigDataCube is an OLAP framework in its early version. The framework does not consider a target data source for materialized data cubes. Instead, it keeps the result of materialized data cube in memory (Spark's DataFrame) and allows interactive access to RESTful services. It does not consider any persistence storage for materialized data cubes. The application of BigDataCube for payment gateway in Digital River was not considered security and data access level grants. It is an important Data Privacy factor, the individuals with correct privileges have access to right level of data over the BigDataCube.

8.6 Future Works

BigDataCube is its initial version as a Distributed OLAP engines. Various features can be introduced to make it a complete Distributed OLAP engines. One important feature is to make materialized cube even fast accessible. In the current version, Materialized Cubes represents cached Spark DataFrame collections. The Data Cube Materialization process can be further extended to write all materialized cubes into a Distributed Multi-index model, which allows even fast access to the Data Cubes.

Visualization of Data Cube via Pivot View does not consider as a major feature in the thesis. But, it is a significant functionality as an analytical tool that can be further improved with numerous front-end features such as Export Data in several formats (e.g Excel, CSV, PDF, etc), User Profiles to share Cubes with other users, User Profiles that grants access to different level of cubes. Pagination that views cubes in pages, etc.

Apache kylin [5] is known as the distributed OLAP engine today. During the thesis, they released a beta version of Apache Kylin, which extends the OLAP engine with partial data cube materialization algorithm [61] based on Spark RDD. It is interesting to compare the performance of Kylin's Data Cube Materialization with BigDataCube. As BigDataCube engine is based on Spark DataFrame, there is a high chance our solution outperforms Spark RDD implementation of Apache Kylin. BigDataCube can be further improved to support many Distributed Data Storage systems, and make it a unique feature among other solutions.

List of Figures

List of Tables

Bibliography

[1] A. Nandi et al. *Data Cube Materialization and Mining over MapReduce*. 2012. URL: http://arnab.org/files/nandi_distributed_cubing.pdf.

[2] Sameet Agarwal et al. "On the Computation of Multidimensional Aggregates". In: *Proceedings of the 22th International Conference on Very Large Data Bases*. VLDB '96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 506–521. ISBN: 1-55860-382-4. URL: http://dl.acm.org/citation.cfm?id=645922.673497.

[3] *Apache Hadoop*. URL: http://hadoop.apache.org/ (visited on 03/02/2017).

[4] *Apache HBase*. URL: http://hbase.apache.org/ (visited on 03/02/2017).

[5] *Apache Kylin*. 2015. URL: http://kylin.apache.org/.

[6] *Apache Spark*. URL: http://spark.apache.org/ (visited on 03/02/2017).

[7] *Apache Storm*. URL: http://storm.apache.org/ (visited on 03/02/2017).

[8] Michael Armbrust et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Tech. rep. UCB/EECS-2009-28. EECS Department, University of California, Berkeley, Feb. 2009. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

[9] Michael Armbrust et al. "Spark SQL: Relational Data Processing in Spark". In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD '15. Melbourne, Victoria, Australia: ACM, 2015, pp. 1383–1394. ISBN: 978-1-4503-2758-9. DOI: 10.1145/2723372.2742797. URL: http://doi.acm.org/10.1145/2723372.2742797.

[10] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. "Materialized Views Selection in a Multidimensional Database". In: *Proceedings of the 23rd International Conference on Very Large Data Bases*. VLDB '97. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997, pp. 156–165. ISBN: 1-55860-470-7. URL: http://dl.acm.org/citation.cfm?id=645923.671019.

[11] Daniel Barbará and Mark Sullivan. "Quasi-cubes: Exploiting Approximations in Multidimensional Databases". In: *SIGMOD Rec.* 26.3 (Sept. 1997), pp. 12–17. ISSN: 0163-5808. DOI: 10.1145/262762.262764. URL: http://doi.acm.org/10.1145/262762.262764.

[12] Daniel Barbará and Xintao Wu. "Using Loglinear Models to Compress Datacube". In: *Proceedings of the First International Conference on Web-Age Information Management*. WAIM '00. London, UK, UK: Springer-Verlag, 2000, pp. 311–322. ISBN: 3-540-67627-9. URL: http://dl.acm.org/citation.cfm?id=645939.671372.

[13] Kevin Beyer and Raghu Ramakrishnan. "Bottom-up Computation of Sparse and Iceberg CUBE". In: *SIGMOD Rec.* 28.2 (June 1999), pp. 359–370. ISSN: 0163-5808. DOI: 10.1145/304181.304214. URL: http://doi.acm.org/10.1145/304181.304214.

[14] Bootstrap core team. *Bootstrap*. URL: http://getbootstrap.com/ (visited on 06/01/2017).

[15] Leo Breiman. "Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author)". In: *Statist. Sci.* 16.3 (Aug. 2001), pp. 199–231. DOI: 10.1214/ss/1009213726. URL: http://dx.doi.org/10.1214/ss/1009213726.

[16] Joubert de Castro Lima and Celso Massaki Hirata. *MDAG-Cubing: A Reduced Star-Cubing Approach*. 2007.

[17] Fay Chang et al. "Bigtable: A Distributed Storage System for Structured Data". In: *ACM Trans. Comput. Syst.* 26.2 (June 2008), 4:1–4:26. ISSN: 0734-2071. DOI: 10.1145/1365815.1365816. URL: http://doi.acm.org/10.1145/1365815.1365816.

[18] Y. Chen et al. "PnP: parallel and external memory iceberg cube computation". In: *21st International Conference on Data Engineering (ICDE'05)*. Apr. 2005, pp. 576–577. DOI: 10.1109/ICDE.2005.107.

[19] E. F. Codd. "A Relational Model of Data for Large Shared Data Banks". In: *Commun. ACM* 13.6 (June 1970), pp. 377–387. ISSN: 0001-0782. DOI: 10.1145/362384.362685. URL: http://doi.acm.org.focus.lib.kth.se/10.1145/362384.362685.

[20] Anindya Datta and Helen Thomas. “The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses1”. In: *Decision Support Systems* 27.3 (1999), pp. 289–301. ISSN: 0167-9236. DOI: https://doi.org/10.1016/S0167-9236(99)00052-4. URL: http://www.sciencedirect.com/science/article/pii/S0167923699000524.

[21] Giuseppe DeCandia et al. “Dynamo: Amazon’s Highly Available Key-value Store”. In: *SIGOPS Oper. Syst. Rev.* 41.6 (Oct. 2007), pp. 205–220. ISSN: 0163-5980. DOI: 10.1145/1323293.1294281. URL: http://doi.acm.org/10.1145/1323293.1294281.

[22] Frank Dehne, Todd Eavis, and Andrew Rau-Chaplin. “The cgmCUBE Project: Optimizing Parallel Data Cube Generation for ROLAP”. In: *Distrib. Parallel Databases* 19.1 (Jan. 2006), pp. 29–62. ISSN: 0926-8782. DOI: 10.1007/s10619-006-6575-6. URL: http://dx.doi.org/10.1007/s10619-006-6575-6.

[23] Curtis E. Dyreson. “Information Retrieval from an Incomplete Data Cube”. In: *Proceedings of the 22th International Conference on Very Large Data Bases*. VLDB ’96. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1996, pp. 532–543. ISBN: 1-55860-382-4. URL: http://dl.acm.org/citation.cfm?id=645922.673326.

[24] E. F. Codd, S.B. Codd, and C.T. Salley. *Providing OLAP to User-Analysts: An IT Mandate*. 1993. URL: http://www.minet.uni-jena.de/dbis/lehre/ss2005/sem_dwh/lit/Cod93.pdf.

[25] Raul Estrada and Isaac Ruiz. *Big Data SMACK*. Apress, 2016. ISBN: 9781484221754.

[26] Min Fang et al. “Computing Iceberg Queries Efficiently”. In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. VLDB ’98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 299–310. ISBN: 1-55860-566-5. URL: http://dl.acm.org/citation.cfm?id=645924.671338.

[27] Pedro Furtado and Henrique Madeira. “Data Cube Compression with QuantiCubes”. In: *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery*. DaWaK 2000. London, UK, UK: Springer-Verlag, 2000, pp. 162–167. ISBN: 3-540-67980-4. URL: http://dl.acm.org/citation.cfm?id=646109.679428.

[28] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google File System". In: *SIGOPS Oper. Syst. Rev.* 37.5 (Oct. 2003), pp. 29–43. ISSN: 0163-5980. DOI: 10.1145/1165389.945450. URL: http://doi.acm.org/10.1145/1165389.945450.

[29] Sanjay Goil and Alok Choudhary. "High Performance OLAP and Data Mining on Parallel Computers". In: *Data Mining and Knowledge Discovery* 1.4 (1997), pp. 391–417. ISSN: 1573-756X. DOI: 10.1023/A:1009777418785. URL: http://dx.doi.org/10.1023/A:1009777418785.

[30] Jim Gray et al. "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals". In: *Data Min. Knowl. Discov.* 1.1 (Jan. 1997), pp. 29–53. ISSN: 1384-5810. DOI: 10.1023/A:1009726021843. URL: http://dx.doi.org/10.1023/A:1009726021843.

[31] *Hadoop MapReduce*. URL: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html (visited on 03/02/2017).

[32] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. 3rd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011. ISBN: 0123814790, 9780123814791.

[33] Jiawei Han et al. "Efficient Computation of Iceberg Cubes with Complex Measures". In: *SIGMOD Rec.* 30.2 (May 2001), pp. 1–12. ISSN: 0163-5808. DOI: 10.1145/376284.375664. URL: http://doi.acm.org/10.1145/376284.375664.

[34] Venky Harinarayan, Anand Rajaraman, and Jeffrey D. Ullman. "Implementing Data Cubes Efficiently". In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. SIGMOD '96. Montreal, Quebec, Canada: ACM, 1996, pp. 205–216. ISBN: 0-89791-794-4. DOI: 10.1145/233269.233333. URL: http://doi.acm.org/10.1145/233269.233333.

[35] *HDFS Architecture*. URL: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (visited on 03/02/2017).

[36] Benjamin Hindman et al. "Mesos: A Platform for Fine-grained Resource Sharing in the Data Center". In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. NSDI'11. Boston, MA: USENIX Association, 2011, pp. 295–308. URL: http://dl.acm.org/citation.cfm?id=1972457.1972488.

[37] J. Dean and S. Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*. 2005. URL: http://research.google.com/archive/mapreduce.html.

[38] Jameel Mohammed. *Is Apache Spark going to replace Hadoop*. 2015. URL: http://aptuz.com/blog/is-apache-spark-going-to-replace-hadoop/ (visited on 06/11/2017).

[39] C.S. Jensen, T.B. Pedersen, and C. Thomsen. *Multidimensional Databases and Data Warehousing*. Synthesis lectures on data management. Morgan & Claypool Publishers, 2010. ISBN: 9781608455379. URL: https://books.google.se/books?id=7UJpcKbVcYkC.

[40] Dong Jin et al. "An Incremental Maintenance Scheme of Data Cubes". In: *Proceedings of the 13th International Conference on Database Systems for Advanced Applications*. DASFAA'08. New Delhi, India: Springer-Verlag, 2008, pp. 172–187. ISBN: 3-540-78567-1, 978-3-540-78567-5. URL: http://dl.acm.org/citation.cfm?id=1802514.1802534.

[41] Ruoming Jin et al. "Communication and memory optimal parallel data cube construction". In: *2003 International Conference on Parallel Processing, 2003. Proceedings.* Oct. 2003, pp. 573–580. DOI: 10.1109/ICPP.2003.1240625.

[42] Julian Hyde. *Mondrian Documentation*. 2006. URL: http://mondrian.pentaho.com/documentation/olap.php (visited on 07/21/2017).

[43] Ken Hess. *Hadoop vs. Spark: The New Age of Big Data*. 2016. URL: http://www.datamation.com/data-center/hadoop-vs.-spark-the-new-age-of-big-data.html (visited on 06/11/2017).

[44] Bart Kuijpers and Alejandro A. Vaisman. "A Formal Algebra for OLAP". In: *CoRR* abs/1609.05020 (2016). URL: http://arxiv.org/abs/1609.05020.

[45] Avinash Lakshman and Prashant Malik. "Cassandra: A Decentralized Structured Storage System". In: *SIGOPS Oper. Syst. Rev.* 44.2 (Apr. 2010), pp. 35–40. ISSN: 0163-5980. DOI: 10.1145/1773912.1773922. URL: http://doi.acm.org/10.1145/1773912.1773922.

[46] Laks V. S. Lakshmanan, Jian Pei, and Yan Zhao. "QC-trees: An Efficient Summary Structure for Semantic OLAP". In: *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*. SIGMOD '03. San Diego, California: ACM, 2003, pp. 64–75. ISBN: 1-58113-634-

X. DOI: 10.1145/872757.872768. URL: http://doi.acm.org/10.1145/872757.872768.

[47] Laks V. S. Lakshmanan et al. “Quotient Cube: How to Summarize the Semantics of a Data Cube”. In: 2002, pp. 778–789.

[48] S. Lee, S. Jo, and J. Kim. “MRDataCube: Data cube computation using MapReduce”. In: *2015 International Conference on Big Data and Smart Computing (BIGCOMP)*. Feb. 2015, pp. 95–102. DOI: 10.1109/35021BIGCOMP.2015.7072817.

[49] Lightbend Inc. *Akka HTTP*. URL: http://doc.akka.io/docs/akka-http/10.0.5/scala/http/introduction.html (visited on 06/01/2017).

[50] Marc West. *The mystery of Zipf*. 2008. URL: https://plus.maths.org/content/mystery-zipf (visited on 07/25/2017).

[51] Microsoft System Center. *Understanding OLAP Cubes*. 2016. URL: https://technet.microsoft.com/en-us/library/hh916543(v=sc.12).aspx (visited on 07/21/2017).

[52] Tova Milo and Eyal Altshuler. “An Efficient MapReduce Cube Algorithm for Varied DataDistributions”. In: *Proceedings of the 2016 International Conference on Management of Data*. SIGMOD ’16. San Francisco, California, USA: ACM, 2016, pp. 1151–1165. ISBN: 978-1-4503-3531-7. DOI: 10.1145/2882903.2882922. URL: http://doi.acm.org/10.1145/2882903.2882922.

[53] Oracle. *Oracle OLAP*. URL: http://www.oracle.com/technetwork/database/options/olap/index.html (visited on 07/21/2017).

[54] *Pentaho, A Hitachi Group Company*. URL: http://www.pentaho.com/ (visited on 03/02/2017).

[55] Pivot4J Team. *Pivot4J*. URL: http://www.pivot4j.org/ (visited on 06/01/2017).

[56] Gavin Powell. *Beginning Database Design*. Wiley Publishing, Inc, 2006. ISBN: 9780764574900.

[57] RAML.org. *RAML*. URL: https://raml.org (visited on 06/01/2017).

[58] Thomas A. Runkler. *Data Analytics, Models and Algorithms for Intelligent Data Analysis*. Springer Fachmedien Wiesbaden, 2016. ISBN: 9783658140748.

[59] Saggi Neumann. *Spark vs. Hadoop MapReduce*. 2014. URL: https://www.xplenty.com/blog/2014/11/apache-spark-vs-hadoop-mapreduce/ (visited on 06/11/2017).

[60] Zheng Shao, Jiawei Han, and Dong Xin. “MM-Cubing: computing Iceberg cubes by factorizing the lattice space”. In: *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.* June 2004, pp. 213–222. DOI: 10.1109/SSDM.2004.1311213.

[61] Shaofeng Shi. *By-layer Spark Cubing*. 2017. URL: http://kylin.apache.org/blog/2017/02/23/by-layer-spark-cubing/ (visited on 06/02/2017).

[62] Amit Shukla, Prasad Deshpande, and Jeffrey F. Naughton. “Materialized View Selection for Multidimensional Datasets”. In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. VLDB ’98. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, pp. 488–499. ISBN: 1-55860-566-5. URL: http://dl.acm.org/citation.cfm?id=645924.671189.

[63] Yannis Sismanis et al. “Dwarf: Shrinking the PetaCube”. In: *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’02. Madison, Wisconsin: ACM, 2002, pp. 464–475. ISBN: 1-58113-497-5. DOI: 10.1145/564691.564745. URL: http://doi.acm.org/10.1145/564691.564745.

[64] Yannis Sismanis et al. “Hierarchical Dwarfs for the Rollup Cube”. In: *Proceedings of the 6th ACM International Workshop on Data Warehousing and OLAP*. DOLAP ’03. New Orleans, Louisiana, USA: ACM, 2003, pp. 17–24. ISBN: 1-58113-727-3. DOI: 10.1145/956060.956064. URL: http://doi.acm.org/10.1145/956060.956064.

[65] The Apache Commons Maths. *Commons Math: The Apache Commons Mathematics Library*. 2016. URL: http://commons.apache.org/proper/commons-math/ (visited on 06/11/2017).

[66] The jQuery Foundation. *jQuery, write less, do more.* URL: https://jquery.com/ (visited on 06/01/2017).

[67] *TIBCO Jaspersoft*. URL: https://www.jaspersoft.com (visited on 03/02/2017).

[68] TONBELLER AG. *JPivot*. URL: http://jpivot.sourceforge.net/ (visited on 06/01/2017).

[69] Tatsuo Tsuji, Akihiro Hara, and Ken Higuchi. “An Extendible Multidimensional Array System for MOLAP”. In: *Proceedings of the 2006 ACM Symposium on Applied Computing*. SAC ’06. Dijon, France: ACM, 2006, pp. 503–510. ISBN: 1-59593-108-2. DOI: 10.1145/1141277.1141394. URL: http://doi.acm.org/10.1145/1141277.1141394.

[70] Uniclau S.L. *jbPivot*. URL: http://www.jbpivot.org/ (visited on 06/01/2017).

[71] Vinod Kumar Vavilapalli et al. "Apache Hadoop YARN: Yet Another Resource Negotiator". In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC '13. Santa Clara, California: ACM, 2013, 5:1–5:16. ISBN: 978-1-4503-2428-1. DOI: 10.1145/2523616.2523633. URL: http://doi.acm.org/10.1145/2523616.2523633.

[72] Jeffrey Scott Vitter and Min Wang. "Approximate Computation of Multidimensional Aggregates of Sparse Data Using Wavelets". In: *SIGMOD Rec.* 28.2 (June 1999), pp. 193–204. ISSN: 0163-5808. DOI: 10.1145/304181.304199. URL: http://doi.acm.org/10.1145/304181.304199.

[73] Jeffrey Scott Vitter, Min Wang, and Bala Iyer. "Data Cube Approximation and Histograms via Wavelets". In: *Proceedings of the Seventh International Conference on Information and Knowledge Management*. CIKM '98. Bethesda, Maryland, USA: ACM, 1998, pp. 96–104. ISBN: 1-58113-061-9. DOI: 10.1145/288627.288645. URL: http://doi.acm.org/10.1145/288627.288645.

[74] Wei Wang et al. "Condensed cube: an effective approach to reducing data cube size". In: *Proceedings 18th International Conference on Data Engineering*. 2002, pp. 155–165. DOI: 10.1109/ICDE.2002.994705.

[75] Y. Wang, A. Song, and J. Luo. "A MapReduceMerge-based Data Cube Construction Method". In: *2010 Ninth International Conference on Grid and Cloud Computing*. Nov. 2010, pp. 1–6. DOI: 10.1109/GCC.2010.14.

[76] Dong Xin et al. "C-Cubing: Efficient Computation of Closed Cubes by Aggregation-Based Checking". In: *22nd International Conference on Data Engineering (ICDE'06)*. Apr. 2006, pp. 4–4. DOI: 10.1109/ICDE.2006.31.

[77] Dong Xin et al. "Star-cubing: Computing Iceberg Cubes by Top-down and Bottom-up Integration". In: *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*. VLDB '03. Berlin, Germany: VLDB Endowment, 2003, pp. 476–487. ISBN: 0-12-722442-4. URL: http://dl.acm.org/citation.cfm?id=1315451.1315493.

[78] YAML.org. *YAML*. URL: http://www.yaml.org/about.html (visited on 06/01/2017).

[79] Matei Zaharia et al. “Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing”. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*. NSDI’12. San Jose, CA: USENIX Association, 2012, pp. 2–2. URL: http://dl.acm.org/citation.cfm?id=2228298.2228301.

[80] Yihong Zhao, Prasad M. Deshpande, and Jeffrey F. Naughton. “An Array-based Algorithm for Simultaneous Multidimensional Aggregates”. In: *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*. SIGMOD ’97. Tucson, Arizona, USA: ACM, 1997, pp. 159–170. ISBN: 0-89791-911-4. DOI: 10.1145/253260.253288. URL: http://doi.acm.org/10.1145/253260.253288.

Appendix

Appendix A - Zipf Synthetic Dataset

Zipf Distribution

During 1949, the American linguist George Zipf (1902-1950) explained an interesting feature in languages. As he described, human uses a few number of words very often, and rarely the majority of other words. When the words were ranked in popularity, found that the word in rank 1 was always popular twice as the word in rank 2, and thrice as the word in rank 3. This rank vs. frequency rule was later augmented as Zipf's law. Zipf distribution is in a family of discrete power law probability distributions, which follows many of the real world data distributions.

Apache Commons Math Library

Apache Commons is a well-known open source library for various reusable Java components. Commons Math is a part of the library that bundles with a lightweight, self-contained mathematics and statistics functions. It provides a simple API to generate a dataset that follows Zipf Distribution.

ZipfDistribution class abstracts the utility functions and provides the following method signature for constructing an instance of it.

```
/**
 * Creates a Zipf distribution.
 *
 *
 * @param rng Random number generator.
 * @param numberOfElements Number of elements.
 * @param exponent Exponent.
 *
```

```
 * @exception NotStrictlyPositiveException
 *     if {@code numberOfElements <= 0}
 *     or {@code exponent <= 0}.
 *
 */
ZipfDistribution(RandomGenerator rng,
                     int numberOfElements,
                     double exponent)
```

Generation of Zipf Synthetic Dataset

Creation of sample dataset that follows Zipf distribution is a simple task with Apache Commons Math Library. The following presents the pseudo-code for generating such data collection.

```
*****************************************************
Input Parameters:  numDimensions, numRecords
Returns:    List of DataRecord
*****************************************************

ZipfDistribution zipfDist =
    new ZipfDistribution(new JDKRandomGenerator(),
                              numRecords, 1);

List<DataRecord> dataRecords =
    new ArrayList<DataRecord>()

while x <= numRecords
    DataRecord record = new DataRecord()
    record.id = x

    while y <= numDimensions
        record.values[y] = zipfDist.sample()
    end while

    dataRecords.add(record)

end while
```

Section 5.2.2 uses an implementation of above pseudo code to generate the data collections for the testbed with the following format.

```
DValue1-1 DValue1-4 DValue1-15 DValue1-18 DValue1-21
DValue1-1 DValue1-2 DValue1-13 DValue1-17 DValue1-19
DValue1-1 DValue1-2 DValue1-18 DValue1-22 DValue1-26
DValue1-2 DValue1-8 DValue1-14 DValue1-18 DValue1-30
DValue1-1 DValue1-2 DValue1-19 DValue1-18 DValue1-35
DValue1-1 DValue1-2 DValue1-16 DValue1-10 DValue1-24
DValue1-1 DValue1-3 DValue1-14 DValue1-12 DValue1-32
```

Appendix B - Setting up AWS EC2 Instances

The testbed for experiments described in section 5.2.3 and 7.3.2 was setup on AWS Cloud Environment, as it provides great flexibility in arranging computing and networking resources. Also, their pricing principle "pay-as-you-go" is quite convenient for experiments that require resources for a shorter period.

Amazon Elastic Compute Cloud (Amazon EC2) is the main resource used from AWS Cloud Service for the experiments. AWS EC2 instances can be simply setup with their Web Console in few steps. But setting up a cluster of EC2 instances for Spark, Hadoop or Cassandra require further consideration. With default settings, EC2 instances are assigned with elastic IP addresses, that may change if the EC2 instance was stopped and started again, which causes modifying the configurations for Spark, Hadoop or Cassandra clusters. It can be avoided setting up AWS EC2 instances within Amazon Virtual Private Cloud (Amazon VPC).

AWS VPC

AWS VPC builds a virtual network where AWS EC2 instances can be launched. It provides complete control over IP address range, subnets, route tables and network gateways. Setting up an AWS VPC is a simple task with few steps guide on AWS Console.

VPC with Public and Private Subnets

Log into the AWS Console and select VPC under Networking Content Delivery category. Then select 'Start VPC Wizard' from VPC Dashboard. In the first step, we choose 'VPC with Public and Private Subnet' configuration as in figure 8.1.

A VPC with Public and Private Subnets are more suitable for Spark, Cassandra and Hadoop Cluster setup. That is, we can have entire Cassandra, Hadoop cluster and Spark slaves nodes in Private subnets where instances are closed to a private IP range. In the Public subnet, we can have publically accessible nodes, such as Spark Master, Cassandra CQLSH client nodes, etc. In the next step, we provide a name to VPC and accept default settings to generate the VPC, see figure 8.2.

AWS EC2

EC2 provides resizable compute capacity in AWS Cloud. AWS Console provides various options for selecting Operating Systems, CPU, Memory, Storage and Network Performance for the target EC2 instance.

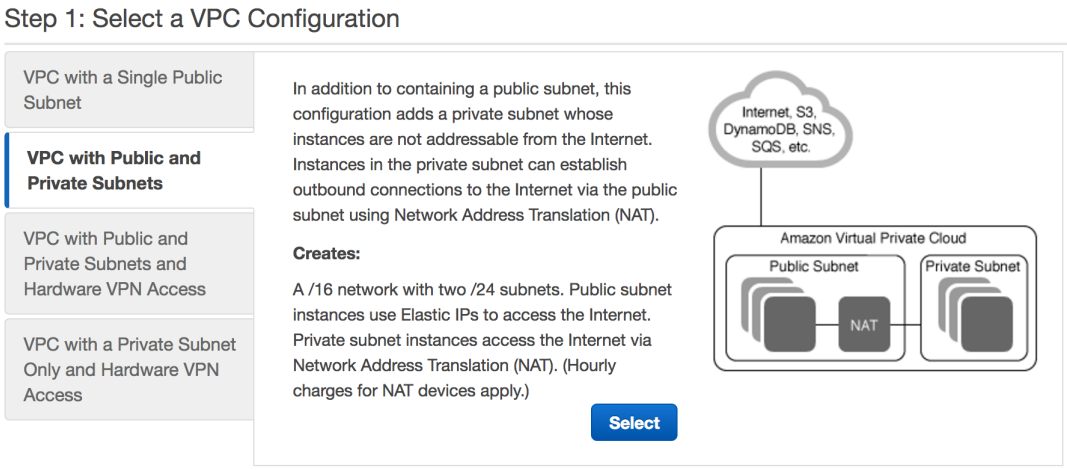


Figure 8.1: AWS VPC, select a VPC with Public and Private Subnets

Figure 8.2: AWS VPC, settings for a VPC with Public and Private Subnets

EC2 Instances on VPC

Setting up EC2 instances on VPC is same as following the EC2 wizard on AWS Console. The only difference is, at Step 3 we select the previously created VPC and subnet, see figure 8.3.

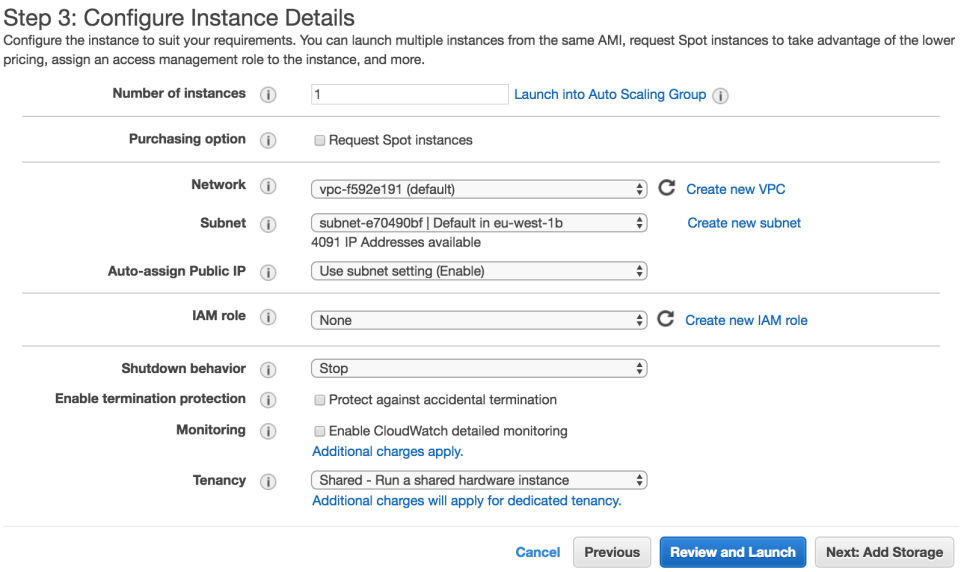


Figure 8.3: AWS EC2, configuration detail

Appendix C - Hadoop Cluster

The execution environment for testbed explained in section 5.2.3 consist of Hadoop Cluster, which was launched on AWS EC2 instances. This guide provides a quick reference for the setup.

Pre-requisites

Create EC2 instances for separate NameNode and DataNodes. In the testbed t2.large instances with Amazon Linux AMI was used. Amazon Linux AMI has already bundled with Open JDK v1.7.0, or we may upgrade it to v1.8.0 with the following command.

```
# Install OpenJdk v1.8.0
sudo yum install java-1.8.0

# Remove existing version 1.7.0
sudo yum remove java-1.7.0-openjdk
```

Download Apache Hadoop version 2.7.3 and extract the bundle in all cluster nodes.

```
# Download Apache Hadoop v2.7.3
wget http://<DownloadMirror>/dist/hadoop/common
           /hadoop-2.7.3/hadoop-2.7.3.tar.gz

# Extract the bundle.
# This will create Hadoop Distribution
# at /home/ec2-user/hadoop-2.7.3
tar -xvf hadoop-2.7.3.tar.gz
```

Setting up Data Nodes

Apply the following configuration file changes in all dedicated Data Nodes.

core-site.xml

```
<!-- Replace 'namenode' with public DNS of Name Node -->
<property>
  <name>fs.defaultFS</name>
```

```
  <value>hdfs://namenode:9000/</value>
</property>
```

hdfs-site.xml

```
<property>
    <name>dfs.replication</name>
    <value>2</value>
</property>
<property>
    <name>dfs.datanode.name.dir</name>
    <value>file:///home/ec2-user/hadoop-2.7.3/data</value>
</property>
```

mapred-site.xml

```
<!-- Replace 'namenode' with public DNS of Name Node -->
<property>
    <name>mapreduce.jobtracker.address</name>
    <value>namenode:54311</value>
</property>
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
```

yarn-site.xml

```
<!-- Replace 'namenode' with public DNS of Name Node -->
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
<property>
    <name>yarn.nodemanager.aux-services
          ...    .mapreduce.shuffle.class</name>
    <value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
```

```
<property>
    <name>yarn.resourcemanager.hostname</name>
    <value>namenode</value>
</property>
```

Setting up Name Node

Apply all above configuration file changes as it is in Data Nodes, except the following Name Node-specific configurations files in etc/hadoop/.

hdfs-site.xml

```
<!-- Replace 'namenode' with public DNS of Name Node -->
<property>
    <name>dfs.replication</name>
    <value>2</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>file:///home/ec2-user/hadoop-2.7.3/data</value>
</property>
```

hadoop-2.7.3/etc/hadoop/masters

```
# Replace 'namenode' with public DNS of Name Node
namenode
```

hadoop-2.7.3/etc/hadoop/slaves

```
# Replace 'datanodeX' with public DNS of each Data Nodes
datanode1
datanode2
.
.
datanodeN
```

Password free SSH

It is important Name Node could access all Data Nodes over SSH without the password. It can be achieved with the following steps.

```
# Create public-private key pair on Name Node
# with empty password
namenode:~ ssh-keygen

# This will generate public key at
# /home/ec2-user/.ssh/id_rsa.pub
#
# Copy this file on all Data Nodes
# and append the file content
# to /home/ec2-user/.ssh/authorized_keys

namenode:~ scp .ssh/id_rsa.pub datanode_ip:.
datanode:~ cat id_rsa.pub >> ~/.ssh/authorized_keys
```

Start Hadoop Cluster

When all configurations are in place, we can start Hadoop cluster from the Name Node with following commands.

```
namenode:~ cd ~
namenode:~ cd hadoop-2.7.3
namenode:~ sbin/start-dfs.sh
namenode:~ sbin/start-yarn.sh
namenode:~ sbin/mr-jobhistory-daemon.sh start historyserver
```

Appendix D - Spark Cluster

The execution environment for testbed explained in section 5.2.3 consist of Spark Cluster in standalone mode, which was launched on AWS EC2 instances. This guide provides a quick reference for the setup.

Pre-requisites

Create EC2 instances for separate Master Node and Slave Nodes. In the testbed t2.large instances with Amazon Linux AMI was used. Amazon Linux AMI has already bundled with Open JDK v1.7.0, or we may upgrade it to v1.8.0 with the following command.

```
# Install OpenJdk v1.8.0
sudo yum install java-1.8.0

# Remove existing version 1.7.0
sudo yum remove java-1.7.0-openjdk
```

Download Apache Spark version 2.1.0 and extract the bundle in all cluster nodes.

```
# Download Apache Spark v2.1.0
wget http://<DownloadMirror>/spark-2.1.0-bin-hadoop2.7.tgz

# Extract the bundle.
# This will create Spark Distribution
# at /home/ec2-user/spark-2.1.0-bin-hadoop2.7
tar -xvf spark-2.1.0-bin-hadoop2.7.taz
```

Setting up Master Node

Setting up Master Node with default setting is straightforward. With default configurations, Spark Master nodes can be started with the following command.
Upon successful setup, Spark Master Web UI is accessible over http://MasterNodeIP:8080.

```
masternode:~ cd ~
masternode:~ cd spark-2.1.0-bin-hadoop2.7
masternode:~ sbin/start-master.sh
```

Setting up Slave Nodes

Setting up Slaves node is also more straightforward as Master node with a single start up script as the following.

```
slavenode:~ cd ~
slavenode:~ cd spark-2.1.0-bin-hadoop2.7
slavenode:~ sbin/start-slave.sh
         ... --master spark://MasterNodeIP:7077
```

Appendix E - Spark vs Hadoop

Result - Data Cube Materialization over Number of Tuples

Table 8.1: Data Cube Materialization over Number of Tuples

	Hadoop MRDataCube	**Spark DataFrame**
1 Million	573 sec	120 sec
2 Millioin	1114 sec	218 sec
3 Millioin	1581 sec	321 sec
4 Millioin	1960 sec	418 sec
5 Millioin	2350 sec	533 sec
6 Millioin	2727 sec	649 sec
7 Millioin	3184 sec	758 sec

Result - Data Cube Materialization over Number of Dimensions

Table 8.2: Data Cube Materialization over Number of Dimensions

	Hadoop MRDataCube	**Spark DataFrame**
3 Dimensions	35 sec	16 sec
4 Dimensions	37 sec	15 sec
5 Dimensions	47 sec	17 sec
6 Dimensions	88 sec	28 sec
7 Dimensions	183 sec	83 sec
8 Dimensions	626 sec	215 sec

Result - Data Cube Materialization over Cluster Size

Table 8.3: Data Cube Materialization over Cluster Size

	Hadoop MRDataCube	**Spark DataFrame**
4 Nodes	2168 sec	828 sec
5 Nodes	2037 sec	665 sec
6 Nodes	2017 sec	554 sec
7 Nodes	1920 sec	485 sec
8 Nodes	1973 sec	416 sec

Appendix F - Spark over Cassandra

Cassandra - Data Cube Materialization over Cluster Size

Table 8.4: Data Cube Materialization over Cassandra Cluster Size

	Spark DataFrame
2 Nodes	169 sec
3 Nodes	135 sec
4 Nodes	112 sec
5 Nodes	110 sec

Appendix G - Spark Cassandra Connector

Bring computation close to the data, is the well-noted phrase in MapReduce. It is achieved with Distributed File System, which partitions large data file across a cluster of computer nodes. Thus, data processing logics in MapReduce model allows distributed data processing close to each data partitions parallelly on a cluster of nodes. Apache Spark brings the same capabilities with more additional features and supports distributed data processing over various Distributed Storage Systems. This guide provides a technical background of Spark-Cassandra connector, how Spark enables distributed data processing over Cassandra.

Cassandra

Figure 8.4 depicts how Cassandra partitions data over the consistent hashing function. The output range of the consistent hashing function is a "ring" or a fixed circular space. Each Node in the Cassandra Cluster is assigned with the key range. In this way, Cassandra allows departure or arrival of new nodes with a minimal impact of data redistribution. Besides, it provides a Data Replication and Membership model to provide an efficient and reliable distributed storage.

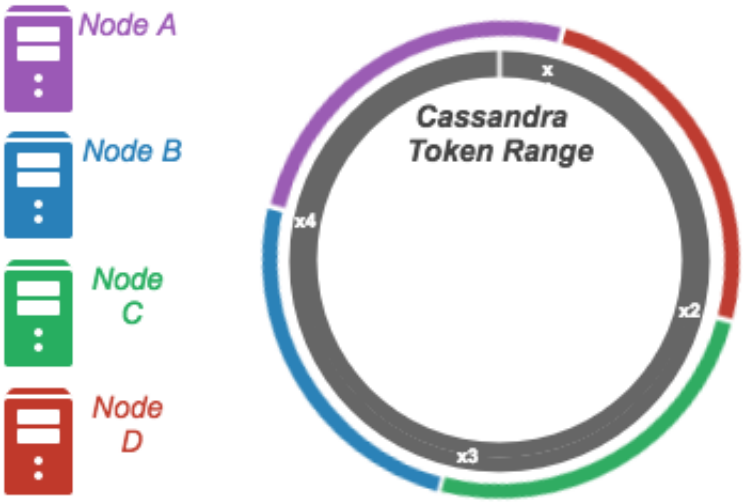


Figure 8.4: Cassandra Token Range

Spark-Cassandra Connector

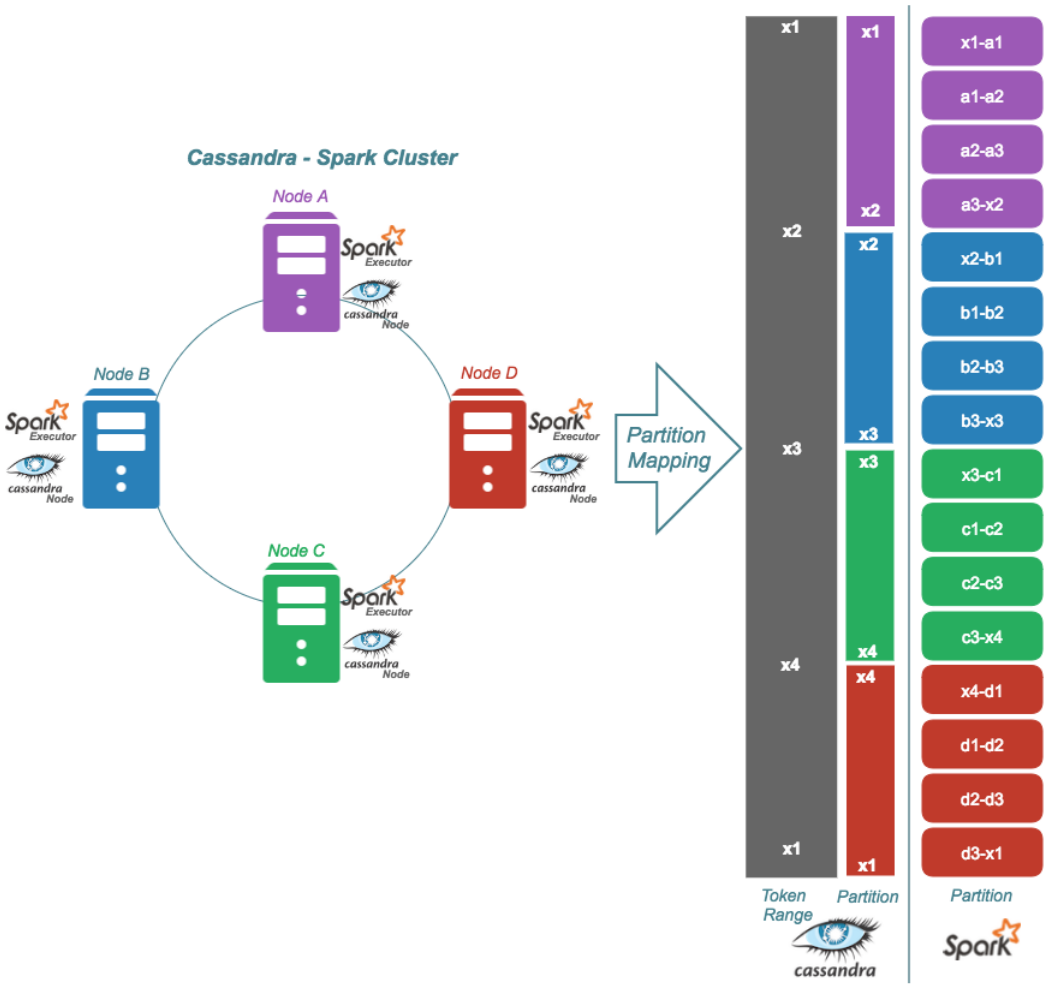


Figure 8.5: Spark Cassandra Partitions

Spark-Cassandra connector provides the RDD implementation for Cassandra distributed storage system. Figure 8.5 illustrates how Spark in-memory data partitions (RDD) are mapped to Cassandra Data partitions. Once the connection to Cassandra is defined, Spark knows the entire token range of the given table and know which node is responsible which token range, and uses that knowledge in assigning executers for Spark jobs.

TRITA TRITA-ICT-EX-2017:125